IPA
INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

15 情経第 1372 号

携帯電話 Java 環境におけるセキュリティ技術に関する調査

2004 年 4 月

独立行政法人 情報処理推進機構

（空白ページ）

目次

1　はじめに.....1
2　背景と目的.....2
3　J2ME 仕様におけるセキュリティ課題.....3
3.1　J2ME セキュリティ概要.....4
3.1.1　J2ME 仮想マシン.....5
3.1.2　J2ME コンフィグレーション.....6
3.1.3　J2ME プロファイル.....7
3.2　J2ME コンフィグレーションのセキュリティ調査.....8
3.2.1　CLDC ライブラリ.....8
3.2.2　CDC ライブラリ.....13
3.2.3　情報資産の保護.....13
3.3　J2ME プロファイルのセキュリティ調査.....14
3.3.1　DoJa ライブラリ.....16
3.3.2　MIDP ライブラリ.....26
3.3.3　KDDI-P ライブラリ.....32
3.3.4　JSCL ライブラリ.....40
3.3.5　情報資産の保護.....44
4　VM のセキュリティモデルとその課題.....46
4.1　KVM のセキュリティ調査.....47
4.1.1　KVM の特徴.....47
4.1.2　CLDC のセキュリティモデル.....47
4.1.3　クラスローダ.....48
4.1.4　クラスファイルベリファイア.....48
4.1.5　アプリケーションの管理.....50
4.1.6　情報資産の保護.....51
4.2　JBlend のセキュリティ調査.....52
4.2.1　JBlend の特徴.....52
4.2.2　Java 仕様との対応.....53
4.2.3　CLDC ライブラリの実装.....54
4.2.4　MIDP ライブラリの実装.....54
4.2.5　DoJa ライブラリの実装.....55
4.2.6　拡張プロファイルの実装.....55
4.2.7　JAM の実装.....56
4.2.8　エミュレータによる検証.....56
4.2.9　情報資産の保護.....74

4.3　JV-Lite2 のセキュリティ調査 75
4.3.1　JV-Lite2 の特徴 75
4.3.2　Java 仕様との対応 77
4.3.3　CLDC ライブラリの実装 77
4.3.4　MIDP ライブラリの実装 77
4.3.5　DoJa ライブラリの実装 78
4.3.6　JAM の実装 78
4.3.7　JV-Lite2 のメモリ構成 81
4.3.8　情報資産の保護 82
5　携帯 Java アプリケーションの運用管理におけるセキュリティ課題 83
5.1　携帯 Java アプリケーションの運用 84
5.1.1　NTT ドコモの運用 86
5.1.2　KDDI(au)の運用 88
5.1.3　vodafone の運用 90
5.2　公開鍵方式によるアプリケーション配布 92
5.3　情報資産の保護 94
6　まとめ 95
6.1　調査結果 95
6.2　考察 96

※ Java™、および Java に関する商標は、米国およびその他の国における米国 Sun Microsystems, Inc.の商標または登録商標です。
※ Sun、Sun Microsystems は、米国およびその他の国における米国 Sun Microsystems, Inc. の商標または登録商標です。
※ ｉモード、ｉアプリ、DoJa、NTT ドコモは株式会社エヌ・ティ・ティ・ドコモの商標または登録商標です。
※ EZ アプリは KDDI 株式会社の登録商標または商標です。
※ vodafone、V アプリは、Vodafone Group Plc またはボーダフォン株式会社の登録商標または商標です。
※ JBlend および JBlend に関連する商標は、日本およびその他の国における株式会社アプリックスの商標または登録商標です。
※ JV-Lite、Compact NetFront、ACCESS は、日本およびその他の国における株式会社 ACCESS の商標または登録商標です。
※ その他、本報告書中に記載の会社名、製品名、サービス名は、各社の商標または登録商標です。なお、文中では™、®マークをすべてに明記していません。

1 はじめに

本報告書は、携帯電話端末の高機能化のけん引役ともなっている携帯電話用 Java について、セキュリティの観点から調査・分析を行い、その結果を報告するものである。

調査は、主として下記の 3 項目について、現在入手可能な公開されている情報をもとに行っている。

- J2ME 仕様におけるセキュリティ課題
- Java バーチャルマシン(VM)のセキュリティモデルおよび機能
- 携帯 Java アプリケーションの運用管理におけるセキュリティ課題

なお、VM については、一部入手できたものを使って簡単な実験も行っている。

また、運用管理に関しては、主として Java アプリケーションの配布と管理について調査した。

2 背景と目的

今日、携帯電話端末は、Java アプリケーションの動作環境を備える等の高機能化を続けており、それらを活用したさまざまなサービスが提供されている。携帯電話端末におけるセキュリティ課題としては、登録されたメールアドレス・電話番号等の情報資産の保護がある。さらに、高機能化や新サービスの提供によって、個人認証のための情報や決済に関係する情報等、保護対象とすべき情報が今後ますます増えることが予想できる。

以上のような背景から、本調査報告は、携帯電話端末におけるセキュリティ設計の観点から、携帯電話用 Java の動作環境におけるセキュリティ確保の状況について報告し、携帯電話端末を使ったアプリケーション開発の場面等で、セキュリティに関する危険を事前に回避するための参考情報とすることを目的としている。

3 J2ME 仕様におけるセキュリティ課題

J2ME 仕様は、携帯電話用 Java に用いられている Java2 プラットフォームである。本章では、J2ME のアーキテクチャ、およびそのセキュリティモデルについて説明し、具体的な実装に関して国内通信キャリアの現状を調査する。

J2ME 仕様のセキュリティモデルについては、3 つのレベルで定義されており、それぞれのレベルにおいてセキュリティ設計がなされている。

- 低レベル（仮想マシン）のセキュリティ
  これは、主に Java 仮想マシンを対象としたセキュリティモデルである。このレベルのセキュリティが確保されることにより、Java アプリケーションの実体であるバイトコードの実行時に不正なコードを排除することによって、J2ME 機器のシステムを破壊しないことなどが保証される。
- アプリケーションレベルでのセキュリティ
  これは、主に Java クラスライブラリとして提供されている API、および API フレームワークを対象としたセキュリティモデルである。クラスライブラリとして実装される Java 言語仕様はもちろんのこと、それらを含めた J2ME コンフィグレーションのコア API、その上位に位置する J2ME プロファイル API によりセキュリティ上の保護が図られている。J2ME 機器内にある個人情報を盗み取るといったプライバシーを侵害するような Java アプリケーションやデータを削除するような悪意を持ったアプリケーションは、Java アプリケーションを構成する API 自身を制限、または保護するようなセキュリティモデルが一般的である。
- エンド・ツー・エンドレベルのセキュリティ
  これは、SSL/TLS などによるピア・ツー・ピアのセキュリティはもちろんのこと、Java プログラムの配信に関するような運用レベルを対象とするセキュリティモデルである。公開鍵方式による証明書を利用した、より安全な Java アプリケーションの配布などもこのレベルに含める。

ここでは、上記のうち、アプリケーションレベルのセキュリティを主に取り上げ、仕様上策定されている API およびそれらの各実装を対象として、どのようにして Java アプリケーションのセキュリティが実現されているのかを調査した。

3.1 J2ME セキュリティ概要

Java 2 プラットフォームのアーキテクチャには、サーバ環境向けの Java 2 Platfrom Enterprise Edition (J2EE)、デスクトップ環境向けの Java 2 Platfrom Standard Edition (J2SE)、そして組み込み機器向けの Java 2 Platfrom Micro Edition (J2ME)の 3 つのエディションが Sun Microsystems 社より提供されている。

J2ME 仕様は、ポケットベルなどの小さなものから、携帯電話、スマートフォン、PDA、セットトップボックスなど、デスクトップコンピュータ程度の機能のある機器までを広くカバーする、リソース制限のある携帯電話端末や埋め込みデバイス向けのものである。

リソースが制限された携帯電話端末などにおいても、他のエディションと同様に第三者が作成した任意の Java アプリケーションをネットワークからダウンロードして実行できるため、汎用のアプリケーションプロセッサとして活用することも可能である。しかし、第三者が作成した Java アプリケーションは、携帯電話端末内の個人情報を持ち出すような不正なアプリケーションである可能性もある。

また、Java アプリケーション配布用のサーバの運用および管理によっては、Java アプリケーションに対する改ざん等により悪意のある不正な Java アプリケーションがサーバ上に公開されたままになることもある。このように信頼できないネットワーク越しにダウンロードした信頼できない Java アプリケーションを制限されたリソースでいかにして安全に実行させるかが J2ME のセキュリティ課題である。

以前より Java の実行環境におけるセキュリティとしては、ネットワーク越しにアプレットをダウンロードして安全に実行するいわゆるサンドボックスと呼ばれるセキュリティモデルがある。また、J2SE などの Java2 プラットフォームでは、セキュリティマネージャなどのセキュリティ関連の API が実装されており、セキュリティ機能が一層強化されている。現在では、ファイルやデータベースなどのリソースへのアクセス制御を詳細に記述したセキュリティポリシーの作成により、クラスの修飾子と位置によってアクセス許可を設けることができるようにもなっている。

しかし、リソースが限定された携帯電話端末や組み込み機器などに対しては、十分なシステムリソースを持ったデスクトップ向けの上記のセキュリティ機能をすべてそのまま実装することは困難である。そこで、制限されたリソースにおいてもセキュリティが考慮されるように J2ME アーキテクチャ（図 1）の各層に対してセキュリティ設計がなされている。

図 1　J2ME のアーキテクチャ

3.1.1　J2ME 仮想マシン

J2ME 仕様の仮想マシンには、Sun Microsystems 社の KVM (K Virtual Machine)をはじめ、アプリックス社の JBlend、ACCESS 社の JV-Lite2 等がある。仮想マシンは、J2ME 機器が備える機能や性能に応じて拡張され実装されている。たとえば、単に KVM や JBlend といっても携帯電話端末の機種ごとの機能に応じて実装されるため、仮想マシンは全く同じものではない。

J2ME 仮想マシンのセキュリティは、J2ME アーキテクチャにおける低レベルのセキュリティモデルに該当する。このレベルでは、クラスファイルベリファイアと呼ばれるバイトコード検証器が実装されている。

J2ME の仮想マシンのクラスファイルベリファイアは、J2SE などの一般的な Java 仮想マシンとは異なり、コンパイル時と実行時の 2 パス方式にてクラスファイルの検証を行う。最初に、Java ソースファイルをコンパイルした後に、コンパイル環境にてプリベリファイと呼ばれるクラスファイルの事前検証を行う。その後、実際にクラスファイルをネットワークからダウンロードして J2ME の仮想マシン上で実行する際に、クラスファイルの形式なども含めた検証を行う。こうすることで、リソースの制限された J2ME 機器においてもバイトコードの検証が可能になる。ただし、実行時のベリファイに必要な処理時間やメモリを減らすことはできるが、プリベリファイを行うことでクラスファイルが若干大きくなってしまうという欠点もある。

クラスファイルベリファイアについては、「4.1.4 クラスファイルベリファイア」で詳細を説明する。

3.1.2 J2ME コンフィグレーション

J2ME プラットフォームにおける Java 仮想マシン仕様と基本的なクラスライブラリを定めた Java 言語仕様、およびその Java 実行環境のことを J2ME コンフィグレーションと呼ぶ。

J2ME コンフィグレーションとしては、CLDC(Connected, Limited Device Configuration)と CDC(Connected Device Configuration)という 2 つのコンフィグレーションの仕様が JCP (Java Commuity Process)により策定されている。国内通信キャリアの携帯電話端末の Java 実行環境には、すべて CLDC が採用されている。

JCP は、Java の仕様やリファレンス実装を検討するオープンな団体である。JCP で扱われる仕様は、仕様策定プロセスを経て JSR (Java Specification Request)として策定され、JSR 番号により管理されている。

J2ME コンフィグレーションでは、ユーザインタフェースや通信のための機能はサポートされておらず、対象となる組み込み機器に最適な Java 仮想マシンと最低限のコアライブラリのみが定義されている。携帯電話端末などの各組み込み機器に特化した機能を利用するには、コンフィグレーション上に対象とするハードウェアの性能や構成に応じたプロファイルを実装する必要がある。

コンフィグレーションのセキュリティは、低レベル、およびアプリケーションレベルのセキュリティに該当する。Java の言語仕様や、保護された API だけに制限することで、Java アプリケーションの安全性を保証している。サンドボックスセキュリティモデルもコンフィグレーションにおいて実装されている。コンフィグレーションのセキュリティモデルについては、「3.2 J2ME コンフィグレーションのセキュリティ調査」以降で説明する。

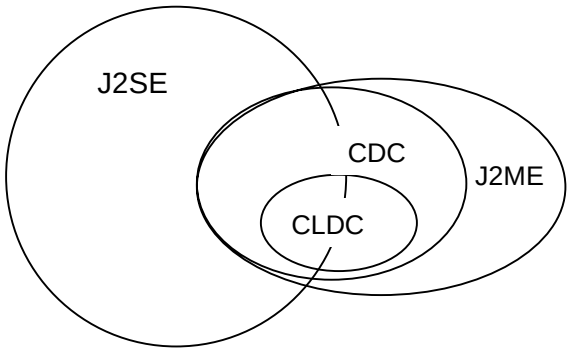

図 2　各仕様の関係

3.1.3　J2ME プロファイル

J2ME プロファイルは、J2ME コンフィグレーションの機能を拡張するクラスライブラリの仕様であり、J2ME コンフィグレーションのコアライブラリ上にアプリケーションレベルのライブラリ仕様を規定するものである。

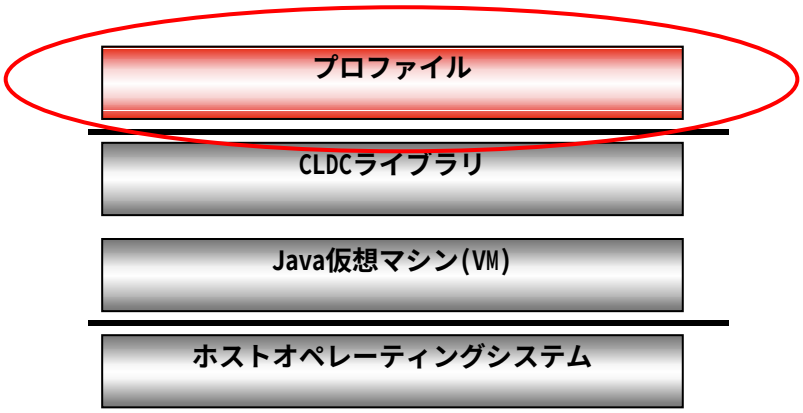

図 3　プロファイルの位置付け

J2ME プロファイルという用語は J2ME コンフィグレーション上のプロファイル一般を指すものであり、たとえば CDC 上のプロファイルも該当する。しかし、国内通信キャリアが実装している携帯電話端末には CLDC を実装したものしかないため、以降では CLDC 上のプロファイルのみを対象とし、単にプロファイルと呼ぶこととする。

プロファイルには、CLDC と同様に JCP により策定された MIDP (Mobile Information Device Profile)と呼ばれる標準のプロファイル仕様がある。また、国内通信キャリアそれぞれが、携帯電話端末に固有のユーザインタフェースやネイティブリソースへのアクセス機能などを実装した独自のプロファイルを提供している。

NTT ドコモでは、MIDP 仕様が JCP で策定される前にすでに DoJa というプロファイルを実装していたため、MIDP とは互換性のない独自のプロファイルが規定されている。また、KDDI(au)と vodafone では JCP で策定された MIDP 1.0 仕様が採用されている。どちらも MIDP が基本のプロファイルとなっているため、両者間の Java アプリケーションの移植は比較的容易であるが、実際には実装の細かな差異により完全な互換性はない。

プロファイルのセキュリティは、J2ME のアプリケーションレベルのセキュリティを担っている。基本的にはコンフィグレーションのコア API を拡張するものであるため、プロファイルにおいてネイティブな機能を直接利用する API 以外は、必ずコンフィグレーションの API を通すことによるセキュリティが確保される。各プロファイルの個々のセキュリティモデルについては、「3.3 J2ME プロファイルのセキュリティ調査」以降で説明する。

3.2 J2ME コンフィグレーションのセキュリティ調査

J2ME コンフィグレーションは、リソースの制限に適合した Java 仮想マシンと最小限のコアライブラリからなる。J2ME アーキテクチャには、対象とするハードウェアの性能やリソースの制限に合わせて、CLDC と CDC の 2 つのコンフィグレーションがある。

CLDC は、16 ビットまたは 32 ビットの CPU と 512K バイト以下のメモリしか搭載していないようなリソースが限られたハードウェアを対象とし、CDC は、32 ビットの CPU と 2M バイト以上のメモリを搭載できるハードウェアを対象とする J2ME コンフィグレーションである。

現在の携帯電話端末のハードウェア制限においては、CLDC コンフィグレーションでしか実装できないが、ハードウェアの制限がクリアされれば、将来的には J2SE のセキュリティモデルとほぼ同等の CDC コンフィグレーションが実装されることになってゆくものと思われる。

低レベルのセキュリティモデルやサンドボックスモデルについては4章にて説明し、本節では、CLDC コンフィグレーションのアプリケーションレベルのセキュリティモデルに着目する。CLDC により提供されている API を利用して、携帯電話端末内のデータを参照したり、破壊したりできないかを調査する。

3.2.1 CLDC ライブラリ

CLDC は、J2ME が対象とするプラットフォームの内、小型の組み込み機器向けに設計されたコンフィグレーションであり、そのリソースでは J2SE と同等の Java 仮想マシンを実装することは難しいため、独自の Java 仮想マシンが定義されている。

CLDC の対象は、以下のような特徴をもつ最小サイズの Java 実行環境を搭載するデバイスである。

- 160K バイトから 512K バイトの合計メモリ
- 16 ビットまたは 32 ビットプロセッサ
- 低電力消費、電池でも動作する
- 特定のネットワークへの接続性、無線、断続的な接続、大域幅の制限がある（多くの場合 9600bps 以下）

CLDC は、小型の組み込みデバイスに搭載する Java の最低限の要件を定義するものであり、Java の仮想マシン仕様および Java の言語仕様や、クラスライブラリも組み込み機器用に最適な Java 実行環境となるよう再定義した仕様である。JCP が策定した CLDC の仕様には CLDC 1.0 と CLDC 1.1 の 2 つのバージョンがあり、現在、国内通信キャリアの携帯電話端末で実装されているのは CLDC 1.0 である。

表 1　CLDC 仕様

仕様	JSR 番号	内容
CLDC 1.0	JSR30	J2ME Connected, Limited Device Configuration
CLDC 1.1	JSR139	Connected Limited Device Configuration 1.1

（1）CLDC 1.0

CLDC 1.0 では、J2SE などの Java 環境とは異なり、対象とするリソースの制限により、Java 仮想マシンおよび言語仕様において以下のような制限がある。

・ 浮動小数点の非サポート
・ ファイナライズの非サポート
・ 弱参照
・ エラー取り扱い上の制限

- 浮動小数点の非サポート
  一般に小型の組み込みデバイスでは、浮動小数点演算を行うためのハードウェアを搭載していないことが多く、メモリ容量や CPU 性能の面でも浮動小数点演算をソフトウェアレベルでエミュレートするのはコストがかかり過ぎる。このため、Java 言語仕様で規定されている float および double は使用できなくなっており、float 型または double 型の引数や戻り値を持つメソッドも削除されている。
- ファイナライズの非サポート
  ファイナライズは、ガベージコレクションがオブジェクトを回収する前に、そのオブジェクトに対して finalize()メソッドを呼び出して何らかの後処理を可能とする仕組みである。ガベージコレクションの際にオブジェクトの状態を管理するには、複雑な状態遷移が必要になる。CLDC では、ガベージコレクションの単純化のために java.lang.Object クラスの finalize()メソッドは削除され、ファイナライズの機構自体がサポートされていない。
- 弱参照
  弱参照を利用すると、オブジェクトが再利用の対象になったとガベージコレクタが判断した際にプログラムは通知を受け取ることができる。ガベージコレクションの単純化のために弱参照はサポートされておらず、java.lang.ref パッケージは提供されていない。
- エラー取り扱い上の制限
  CLDC に搭載される Java 仮想マシンにも J2SE と同様の例外メカニズムが用意されているが、CLDC のターゲットのメモリサイズを考慮して、一部の機能が削除されている。組み込みシステムにおけるエラー状態からの復旧は、通常はデバイス固有の方法で行われる。重大なエラーから復旧しようとするデバイスもあるが、多くの組み込み機器ではエラーに遭遇した際、リソースの制限によりあえて復旧しない場合もある。
  こうした制限から CLDC 1.0 では、システムエラーの際に発生するエラークラスが、java.lang.Error, java.lang.VirtualMachineError, java.lang.OutOfMemoryError の 3 つに制限されている。

また、CLDC においてはセキュリティ上の理由から以下の項目はサポートされていない。

・ JNI(Java Native Interface)
・ ユーザ定義のクラスローダ
・ リフレクション
・ スレッドグループとデーモンスレッド

- JNI
  JNI は、ネイティブなプログラム(関数)を呼び出すための機能である。JNI がサポートされないことにより、携帯電話端末の Java 環境から直接 OS が管理している個人情報などのリソースにアクセスできないことが保証される。
- ユーザ定義のクラスローダ
  ユーザ定義のクラスローダは、ユーザが Java で記述したクラスローダである。ユーザ定義のクラスローダを使用すると、ネットワークプログラムに必要なクラスを必要に応じてロードするといったようなクラスのロード方法を独自に定義できる。
  J2SE では、新たにクラスをロードする必要がある場合は、セキュリティマネージャによりクラスファイルの安全性をチェックするセキュリティモデルになっているが、リソースに制限のある CLDC ではセキュリティマネージャは実装されていない。このため CLDC では、java.lang.ClassLoader クラスが提供されておらず、ユーザ定義のクラスローダを作成できないため、システムクラスが最初にロードされ、オーバーライドされないことが保証される。したがって、外部のネットワークから優先的に読み込むような悪意のある不正クラスローダを含む Java のアプリケーション等は実現不可能である。
- リフレクション
  リフレクション機能がサポートされないことにより、仮想マシン内のクラス、オブジェクト、メソッド、スレッド、その他の実行時構造体の数および内容を走査することはできない。Java プログラムの実行時に動的にクラスを生成したり、クラスやオブジェクトやメソッドなどの内容をチェックしたりするような Java アプリケーションの実現は不可能である。
  また、リフレクションが利用されるオブジェクトの直列化や RMI(Remote Method Invocation)もサポートされていない。
- スレッドグループとデーモンスレッド
  CLDC では、スレッドはサポートされているが、デーモンスレッドとスレッドグループはサポートされていない。J2SE では、セキュリティマネージャにより、スレッドとスレッドグループに適切なアクセス制御を適用する仕組みがある。セキュリティマネージャの働きにより、スレッドやスレッドグループはサンドボックスで安全を確保できる。しかし、CLDC では、リソースの制限によりセキュリティ関連の API やセキュリティマネージャが実装されないため、スレッドグループの機能はサポートされていない。

以降では、具体的な CLDC パッケージ、および API の内、通信関連やファイルなどのリソースに対してアクセスする可能性があるものを対象としてセキュリティ上安全であるかを検討する。CLDC では以下のパッケージが提供されており、javax.microedition.io パッケージ以外は、標準 J2SE パッケージのサブセットとなっている。

表 2　CLDC パッケージ概要

パッケージ名	CLDC 1.0	CLDC 1.1	概要
java.io	○	○	入出力
java.lang	○	○	言語基本
java.lang.ref	×	○	弱参照
java.util	○	○	ユーティリティ
javax.microedition.io	○	○	汎用フレームワーク

- java.io パッケージ

J2SE の java.io パッケージの内､ファイル操作に関連する以下のクラスが存在しないため、ファイルを改ざんしたり、ファイルに感染したりする不正プログラムを作成できず、CLDC 環境では動作させることができない。

表 3　java.io パッケージ概要

クラス名	概要
File	ファイルおよびディレクトリのパス名の抽象表現
FileDescriptor	ファイル記述子クラスのインスタンスは、開いたファイル、開いたソケット、またはバイトの別のソース (シンク) を表す、基本となるマシン固有の構造への不透明なハンドルとして機能
FileInputStream	ファイルシステムのファイルから入力バイトを取得
FileOutputStream	File または FileDescriptor にデータを書き込むためのファイル出力ストリーム
FilePermission	ファイルまたはディレクトリへのアクセスクラス
FileReader	文字ファイルからの読み込みのための簡易クラス
FileWriter	文字ファイルを書き込むための簡易クラス
FilterInputStream	ほかの入力ストリームを格納し、それをデータの基本的なソースとして使用して、データを途中で変換および追加する機能を提供
FilterOutputStream	出力ストリームをフィルタ処理するすべてのクラスのスーパークラス
FilterReader	フィルタ処理された文字列ストリームを読み込むための抽象クラス
FilterWriter	フィルタ処理された文字列ストリームを書き込むための抽象クラス
RandomAccessFile	ランダムアクセスファイルからの読み込み、書き込みの両方をサポートするクラス

- java.lang パッケージ

Object クラスから finalize、clone メソッドが削除されている。Runtime クラスは存在するが、外部プログラムを実行する exec メソッド、すなわち、外部プログラムの実行関数がサポートされておらず、この機能も動作しないことが保証される。また、CLDC の制限により以下のクラスも削除されている。

表 4　java.lang パッケージ内の削除クラス

クラス名	概要
ClassLoader	ユーザ定義のクラスローダ
RuntimePermission	実行時のアクセス権関連
Package	クラスローダによって取り出されるパッケージの実装および仕様についてのバージョン情報を保持
SecurityManager	アプリケーションでセキュリティポリシーを実装できるようにするクラス
ThreadGroup	スレッドグループ

- javax.microedition.io パッケージ

ネットワークのインタフェースを汎用化して定義した通信接続のフレームワークであり、各プロトコルの実装は上位のプロファイルで定義される。CLDC が規定される際に新たに追加されたパッケージであるが、Generic Connection Framework(GCF)と呼ばれる汎用のインターフェースフレームワークである。

Connector クラスの open スタティックメソッドにより、ネイティブで実装された Connection インタフェースを持つ実装オブジェクトが利用できる。Connector の実装は、各携帯電話端末の機種に依存し、プロファイルで提供されるインタフェースのセキュリティについては実装依存となる。

（2）CLDC 1.1

CLDC 1.1 では CLDC 1.0 と比べて新たに以下の点が変更された。

・「Float」「Double」のクラスが追加、浮動小数点演算のサポート
・弱参照の追加
・「Calendar」「Date」「Timezone」クラスの再設計
・「NoClassFoundError」クラスの追加とエラーハンドリングの機能改良
・クラスライブラリの一部のバグ修正
・浮動小数点演算のサポートにより、最小メモリ合計を 160K から 192K に変更

サポートする組み込み機器の性能には変化があったが、セキュリティモデルについてはバージョン間に変化がなく CLDC1.0 と同等である。

3.2.2 CDC ライブラリ

CDC は、CLDC のスーパーセットであり、カーナビ、POS 端末、TV セットトップボックス、PDA、ページャ、スマートフォンなどの比較的ハードウェアの制約が少ないプラットフォームを対象とする組み込みデバイス向けに設計された。CDC 対応デバイスは、ハードウェアの物理的制限が少ないので、標準的な Java の実行環境に近い。CDC で使用される CVM と呼ばれる Java 仮想マシンは、Java2 の Java 仮想マシン仕様をほぼ実装したものであり、Syncronization の高速化やクラスの ROM 化、ネイティブスレッドのサポートなど、組み込み向けにコンパクトに作られてはいるが、多くの部分で J2SE との互換性がある。

CDC の対象は、一般的に以下のような特徴をもつ Java 実行環境を搭載する機器である。

- 2M バイト以上の合計メモリ
- 32 ビットプロセッサ
- 低電力消費、電池でも動作する
- 特定のネットワークへの接続性

JCP が策定した CDC の仕様には、CDC-1.0 と CDC-1.1（策定中）の 2 つのバージョンがある。

表 5　CDC 仕様

仕様	JSR 番号	内容
CDC-1.0	JSR36	J2ME Connected Device Configuration
CDC-1.1	JSR218	J2ME CDC 1.1

本文では、携帯電話端末の Java 環境を主題としているので、CDC についてはこれ以上触れないこととする。

しかし、ハードウェアの進歩により、携帯電話端末のバッテリ容量の問題などが解決されれば、明らかにコンフィグレーションは CLDC から CDC に移行してゆくであろうと思われる。そうした場合 J2ME のセキュリティモデルには、J2SE と同等のセキュリティモデルが適用できるため、セキュリティ関連の API も提供され、より柔軟に Java アプリケーションを作成、配布できるようになるものと思われる。

3.2.3 情報資産の保護

国内通信キャリアの携帯電話端末の Java 実行環境として現在採用されている CLDC コンフィグレーションは、低レベルのセキュリティとして、サンドボックスモデルをそのまま継承している。また、アプリケーションレベルでのセキュリティは、CLDC 1.0、CLDC 1.1 のどちらのコンフィグレーションにおいても、ファイルなどのリソースにアクセスしたり改ざんしたりすることはできない。そのため、コンフィグレーションにおいては、情報資産は保護されており安全である。

3.3 J2ME プロファイルのセキュリティ調査

現在、国内通信キャリアの Java アプリケーションのダウンロードサービスには、NTT ドコモの i アプリ、KDDI(au)の EZ アプリ、vodafone の V アプリがある。携帯電話端末の Java 環境には、それぞれ以下のプロファイルが実装されている。以降では、便宜的に上記の Java アプリケーションのダウンロードサービス名を国内通信キャリアでの Java アプリケーションを特定する名前としても扱うものとする。

表 6　各国内通信キャリアの実装プロファイル

国内通信キャリア	プロファイル	拡張プロファイル
NTT ドコモ	DoJa（i アプリ基本）	i アプリオプション、i アプリ拡張
KDDI(au)	MIDP	KDDI-P(KDDI-Profile)
vodafone	MIDP	JSCL(J-PHONE Specific Class Library)

DoJa は、NTT ドコモにより提供されている i モード端末向けのプロファイルである。これは i アプリと呼ばれる Java アプリケーションを開発するために NTT ドコモが独自に策定したプラットフォームであり、NTT ドコモの携帯電話端末の機種に応じて DoJa-1.0、DoJa-2.0、DoJa-3.0 のバージョンが実装されている。

MIDP は、JCP で策定された CLDC 上の標準プロファイルであり、KDDI(au)と vodafone に採用されている。MIDP 仕様には MIDP 1.0、MIDP 2.0 のバージョンがあり、MIDP プロファイルを利用した端末機器上のアプリケーションを MIDlet と呼ぶ。

KDDI-P は、KDDI(au)の携帯電話端末独自の拡張ライブラリであり、JSCL は、vodafone の携帯電話端末独自の拡張ライブラリである。それぞれの機種に応じた拡張ライブラリにもバージョンがある。

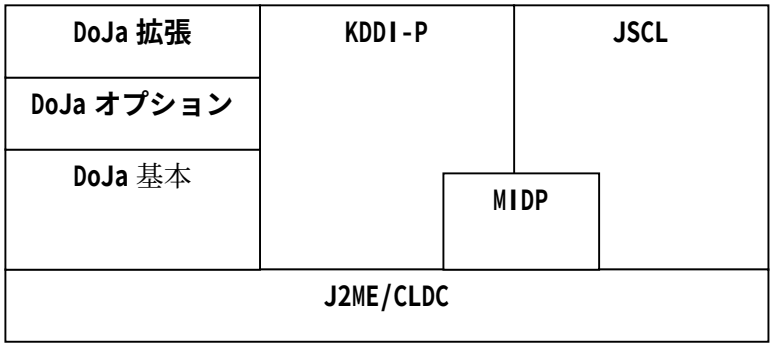

図 4　DoJa と MIDP

本節では、J2ME アーキテクチャのプロファイルのレベルに焦点を当て、J2ME のアプリケーションレベルのセキュリティモデルを検討する。各種プロファイルで提供されている API を参照し、通信関連の API や携帯電話端末のネイティブなリソースにアクセスする API が実装されているかどうかを調べる。その結果、携帯電話端末に登録されたメールアドレス

や電話番号などの情報資産を参照したり、破壊したりすることが可能であるのかを検討する。

以降では、各プロファイルについて通信関連の API やネイティブなリソースにアクセスする API の概要を拾い出し、以下のような形式で記述する。これは、大まかにどのような機能があるかを示すものであり、クラス変数やインスタンス変数、その記述内容の詳細については省略している。

表 7　クラス・インタフェース概要の記述例

クラス名、またはインタフェース名 　　クラスまたはインタフェースの概要
クラス階層、またはインタフェース階層
メソッド 1 のシグネチャ 　　メソッドの概要
・ ・ ・
メソッド n のシグネチャ 　　メソッドの概要

3.3.1 DoJa ライブラリ

NTT ドコモから提供されている DoJa ライブラリには、携帯電話端末の機種に応じて 1.0、2.0、3.0 の 3 つのバージョンが存在する。機種が新しくなるたびにライブラリがバージョンアップされており、アプリケーションレベルのセキュリティ機能も、バージョン毎に少しずつ異なっている。

表 8　DoJa のバージョン一覧

バージョン	機種	発表時期
DoJa-1.0	503i シリーズ向け	2001 年 1 月
DoJa-2.0	504i シリーズ向け	2002 年 5 月
DoJa-3.0	505i シリーズ向け	2003 年 5 月

DoJa と呼ばれる i アプリ API は、以下の 3 つのカテゴリの API 群から構成されている。i アプリオプション API や i アプリ拡張 API には、将来の DoJa プロファイルにおいて i アプリ基本 API に取り入れられる可能性のあるものや、将来のプロファイルで削除される可能性のあるものも含まれている。

（1） i アプリ基本 API
共通仕様として API および動作が規定されており、全ての機種に共通に搭載される API。

（2） i アプリオプション API
共通仕様として API および動作が規定されているが、搭載有無の判断はメーカに委ねられる API。このカテゴリの API は搭載されていない機種がある。

（3） i アプリ拡張 API
同一の機能を実現するものであっても、API および動作がメーカ毎に異なる可能性のある API。このカテゴリの API は搭載されていない機種があることと、機種により API の使用方法などが異なる（同機能を持つ機種間であってもソースコードレベルでの互換性が保持されない）ことがある。

i アプリは通常、メーカを限定せずに動作できるよう考慮し、CLDC のコア API と i アプリ基本 API だけを利用して作成されることが多い。

表 9　DoJa パッケージ概要

DoJa パッケージ名	DoJa-1.0	DoJa-2.0	DoJa-3.0	概要
com.nttdocomo.device	×	×	○	カメラや赤外線リモコンなどのデバイス制御
com.nttdocomo.io	○	○	○	汎用コネクションフレームワーク入出力
com.nttdocomo.lang	○	○	○	プリミティブ
com.nttdocomo.net	○	○	○	ネットワーク通信
com.nttdocomo.system	×	×	○	電話帳機能やメール機能などの端末ネイティブ利用
com.nttdocomo.ui	○	○	○	アプリケーションおよびユーザインタフェース
com.nttdocomo.util	○	○	○	ユーティリティ
com.nttdocomo.opt.device	×	×	○	拡張デバイス制御
com.nttdocomo.opt.ui	×	○	○	拡張ユーザインタフェース
com.nttdocomo.opt.ui.j3d	×	○	○	拡張 3D グラフィックス 1
com.nttdocomo.opt.ui.j3d2	×	○	○	拡張 3D グラフィックス 2

i アプリ対応携帯電話端末のセキュリティを確保するための Java セキュリティモデルは、以下のセキュリティ機能によって実現されている。

表 10　DoJa のセキュリティ機能

項番	セキュリティ機能	DoJa-1.0	DoJa-2.0	DoJa-3.0
［1］	アプリケーションの排他実行	○	○	○
［2］	JAM によるアプリケーション管理	○	○	○
［3］	データ領域のアクセス制限	○	○	○
［4］	ネイティブデータへのアクセス制限	○	○	○
［5］	HTTPS プロトコルの実装	○	○	○
［6］	ネイティブアプリケーションとの連携	×	○	○
［7］	トラステッド i アプリ	×	×	○

［1］アプリケーションの排他実行

携帯電話上では同時に 1 つのアプリケーションしか動作できない。したがって、あるアプリケーションが別のアプリケーションに干渉したり、別のアプリケーションのデータにアクセスしたりすることはできない。ただし、KVM はマルチスレッドをサポートしており、1 つのアプリケーションの中で通信とユーザインタフェース処理を並行して行うといったマルチスレッドプログラミングは可能である。

［2］JAM によるアプリケーション管理

JAM はアプリケーションのリストアップ、追加、削除などの管理と KVM へのロード（プログラムの読込み）を行う独立したネイティブコンポーネントであり、Java アプリケーションからはこれを制御することはできない。Java アプリケーションと JAM ファイルの両方が揃って正当であることを確認してから JAM は実行を許可するため、ダウンロードしたアプリケーションの不正な行為やバグによる誤動作などから保護することができる。

［3］データ領域のアクセス制限

Java アプリケーションは、携帯電話の記憶装置（スクラッチパッド）に制限されたアクセスしかできない。あるアプリケーションに割り当てられたスクラッチパッドの領域に、他のアプリケーションからアクセスすることは許可されない。したがってダウンロードされたアプリケーションが、他のアプリケーションのためにスクラッチパッドに保存された個人情報などをユーザの知らないうちに読み込んで別のサーバに送信するなどの不正な行為を行うことはできない。

［4］ネイティブデータへのアクセス制限

Java アプリケーションがメモリダイヤルなどの個人情報を含んだネイティブデータにアクセスすることはできない。また、ブラウザやダイヤラーなどのネイティブアプリケーションにアクセスする機能では、ユーザの意思に反して Web ブラウジングや音声通話発信を行うことのないよう、API 呼び出し時に必ずユーザに動作への同意を求める実装になっている。

［5］HTTPS プロトコルの実装

i アプリ対応携帯電話で使用できるネットワークプロトコルは HTTP、HTTPS である。i アプリ対応携帯電話では、携帯電話側にて SSL（Secure Socket Layer）によるサーバ認証をサポートしている。ｉモードの通信機能と SSL サポートを利用することで、HTTPS による通信を行うことができる。安全な接続が必要なクライアント・サーバーアプリケーションでは、HTTPS プロトコルを利用する。HTTPS サーバに接続するクライアントプログラムのコードは、URL 文字列のプロトコルセクションに "http" の代りに "https" と指定する以外、HTTP プロトコルの場合と全く同じである。

HttpConnection に渡される URL はアプリケーションが最初にダウンロードされたときの URL と同じプロトコル、ホスト、ポート番号が指定されていなければならない。したがって、HTTPS を介してダウンロードされたアプリケーションでなければ HTTPS プロトコルを使用することはできない。また、HTTPS を介してダウンロードされたアプリケーションは、HTTP プロトコルは使用できない。

Java アプリケーションは、そのアプリケーション自身のダウンロード元であるサーバとしか通信できない。このネットワークセキュリティ機能により、Java アプリケーションが、ユーザの知らない他のサーバに対して情報を送信するようなことはない。

表 11　HttpConnection インタフェース概要

HttpConnection 　　HTTP コネクションのインタフェース
interface javax.microedition.io.Connection 　　interface javax.microedition.io.InputConnection 　　interface javax.microedition.io.OutputConnection 　　　　interface javax.microedition.io.StreamConnection 　　　　　　interface javax.microedition.io.ContentConnection 　　　　　　　　interface com.nttdocomo.io.HttpConnection
public void connect() 　　HTTP で接続し、メッセージを送受信する。
public void close() 　　接続を閉じてリソースを解放する。
public InputStream openInputStream() 　　入力ストリームを取得する。
public OutputStream openOutputStream() 　　出力ストリームを取得する。
public String getEncoding() 　　コンテントのエンコード形式の識別文字列を取得する。
public long getLength() 　　コンテンツの長さを取得する。
public String getType() 　　コンテントのタイプの識別文字列を取得する。
public String getURL() 　　接続先の URL を取得する。
public void setRequestMethod(String method) 　　リクエストメソッドを設定する。
public void setRequestProperty(String key, String value) 　　ヘッダのプロパティ値を設定する。
public int getResponseCode() 　　レスポンスコードを取得する。
public String getResponseMessage() 　　レスポンスメッセージを取得する。
public String getHeaderField(String name) 　　ヘッダ情報を取得する。
public long getDate() 　　メッセージの日付を取得する。
public long getExpiration() 　　メッセージの有効期限を取得する。
public long getLastModified() 　　メッセージの更新時刻を取得する。
public void setIfModifiedSince(long ifmodifiedsince) 　　"If-Modified-Since"ヘッダの値を設定する。

［6］ネイティブアプリケーションとの連携

DoJa-1.0 プロファイルでは、セキュリティ確保のため Java アプリケーションと他のネイティブアプリケーションの連携動作をサポートしていなかったが、DoJa-2.0 からは、携帯電話の各リソースへのアクセスが緩和された。DoJa-2.0 では、アプリケーションと携帯電話に搭載されているネイティブアプリケーション間の連携機能が強化され、以下のことができるようになった。

（1） 赤外線通信
（2） i アプリからの Web ブラウザ起動
（3） i アプリからの音声発信（電話帳に登録したデータは見えない）
（4） Web ブラウザからの i アプリ起動
（5） メールからの i アプリ起動
（6） 赤外線ポートからの i アプリ起動
（7） カメラ起動と、撮影画像のスクラッチパッドへの保存

ただし、DoJa-1.0 と同様にダウンロード元以外との通信は行えず、個人情報が漏れる可能性のあるメールやアドレス帳へのアクセスもできない。また、連携動作が行われる際には必ずユーザの同意を得るという機構のため、ユーザにとって安全なネイティブアプリケーション連携が可能になっている。

表 12　ApplicationStore クラス概要

ApplicationStore 携帯電話のネイティブの i アプリ管理機能にアクセスする
java.lang.Object com.nttdocomo.system.ApplicationStore
public static ApplicationStore selectEntry() ユーザ操作により i アプリデータのエントリを取得する。
public int getId() i アプリデータのエントリの ID を取得する。

表 13　Bookmark クラス概要

Bookmark 携帯電話のネイティブのブックマーク機能にアクセスする
java.lang.Object com.nttdocomo.system.Bookmark
public static int addEntry(String url, String title) ユーザ操作によりブックマークエントリを新規登録する。

表 14　CallRecord クラス概要

CallRecord 　　携帯電話のネイティブの音声発着信履歴機能にアクセスする
java.lang.Object 　　com.nttdocomo.system.CallRecord
public static CallRecord getLastRecord(int type) 　　発着信履歴の最新のエントリを取得する。
public int[][] getPhoneBookID() 　　相手の電話帳エントリ ID を取得する。
public XString getPhoneNumber() 　　相手の電話番号(XString)を取得する。
public XString getDateString(String pattern) 　　発着信日時(XString)を取得する。
public java.lang.Boolean isSucceeded() 　　接続が成功したか否かを取得。

表 15　MailAgent クラス概要

MailAgent 　　携帯電話のネイティブのメール機能にアクセスする
java.lang.Object 　　com.nttdocomo.system.MailAgent
public static Mail getLastIncoming() 　　最新未読メールに対応する Mail オブジェクトを取得する。
public static boolean send(String subject, String[] addresses, String body) 　　ユーザ操作によりメールを送信する。
public static boolean send(String subject, XString address, String body) 　　ユーザ操作によりメールを送信する。
public static boolean send(MailDraft mail) 　　ユーザ操作によりメールを送信する。

表 16　MessageAgent クラス概要

MessageAgent 　　メッセージ i アプリのネイティブ機能にアクセスする
java.lang.Object 　　com.nttdocomo.system.MessageAgent
public static int size(int type, boolean unseen) 　　メッセージフォルダ中のメッセージの数を取得する。
public static int[] getIds(int type, boolean unseen) 　　メッセージフォルダ中のメッセージの ID を取得する。
public static Message getMessage(int type, int id) 　　メッセージを取得する。
public static boolean send(String subject, String[] addresses, String body, byte[] data) 　　メッセージを送信する。
public static boolean send(String subject, XString address, String body, byte[] data) 　　メッセージを送信する。
public static boolean send(MessageDraft message) 　　メッセージを送信する。
public static boolean send(MessageSent message) 　　メッセージを再送信する。
public static void delete(int type, int id) 　　メッセージを削除する。
public static void setSeen(int id, boolean seen) 　　受信フォルダのメッセージの既読・未読情報を設定する。
public static boolean isSeen(int id) 　　受信メッセージの既読・未読情報を取得する。

表 17　Phone クラス概要

Phone 　　携帯電話のネイティブの音声通話機能にアクセスする
java.lang.Object 　　com.nttdocomo.util.Phone
public static final void call(String phoneNumber) 　　音声発信機能を呼び出す。ユーザ確認のダイアログが表示される。
public static final void call(XString phoneNumber) 　　音声発信機能を呼び出す。
public static final String getProperty(String key) 　　指定されたキーに該当するプロパティ値を取得する。取得できるプロパティのキーは、"terminal-id"と"user-id"のみ。

表 18　PhoneBook クラス概要

PhoneBook 携帯電話のネイティブの電話帳機能にアクセスする
java.lang.Object com.nttdocomo.system.PhoneBook
public static int[] addEntry(String name, String kana, String[] phoneNumbers, String[] mailAddresses, String groupName) ユーザ操作により電話帳エントリを新規登録する。
public static int[] addEntry(String name, String kana, String[] phoneNumbers, String[] mailAddresses, int groupId) ユーザ操作により電話帳エントリを新規登録する。
public static int[] addEntry(PhoneBookParam param) ユーザ操作により電話帳エントリを新規登録。
public int getId() 電話帳エントリの ID を取得する。
public XString getName() 電話帳エントリの名前を取得する。
public XString getName(int part) 電話帳エントリの姓や名を取得する。
public XString getKana() 電話帳エントリの読み仮名を取得する。
public XString getKana(int part) 電話帳エントリの姓や名の読み仮名を取得する。
public XString[] getPhoneNumbers() 電話帳エントリの電話番号を取得する。
public XString getPhoneNumber(int index) 電話帳エントリの電話番号の 1 つを取得する。
public XString[] getMailAddresses(int part) 電話帳エントリのメールアドレスを取得する。
public XString getMailAddress(int index, int part) 電話帳エントリのメールアドレスの 1 つを取得する。
public XString getGroupName() 電話帳エントリのグループ名を取得する。
public int getGroupId() 電話帳エントリのグループ ID を取得する。

表 19　PhoneBookGroup クラス概要

PhoneBookGroup 携帯電話のネイティブの電話帳グループ機能にアクセスする
java.lang.Object com.nttdocomo.system.PhoneBookGroup
public static PhoneBookGroup selectEntry() ユーザ操作により電話帳グループのエントリを取得する。
public static PhoneBookGroup getEntry(int id) 電話帳グループの ID を指定して、ユーザ操作なしに電話帳グループのエントリを取得する。
public static int addEntry(String name) ユーザ操作により電話帳グループを新規登録する。
public int getId() 電話帳グループの ID を取得する。
public XString getName() 電話帳グループの名前を取得する。

表 20　Schedule クラス概要

Schedule 携帯電話のネイティブのスケジューラ機能にアクセス
java.lang.Object com.nttdocomo.system.Schedule
public static int getSupportedTypes() サポートしているスケジュール時刻のタイプを取得する。
public static boolean addEntry(String description, ScheduleDate date, boolean alarm) ユーザ操作によりスケジュールを新規登録する。

［7］トラステッド i アプリ

DoJa-3.0 では、セキュリティ機能を緩和するトラステッド i アプリが導入された。トラステッド i アプリとは NTT ドコモに認定された i アプリであり、携帯電話本体に登録した電話帳・発着信履歴などとの連携や、i アプリをダウンロードしたサーバ以外との通信、i モードメールとの連携といった従来の i アプリでは許可されなかった操作が可能となる。トラステッド i アプリでは、以下のようなことが行える。

（1）　メール送信
（2）　受信メールのアプリケーション内での利用
（3）　電話帳の新規登録
（4）　電話帳の参照
（5）　画像のメモリ保存
（6）　保存されている画像の呼び出し

ただし、無条件にこれらを許すと、電話帳の内容を取得したり、電話帳に勝手に電話番号を登録したりする悪意のある i アプリが作成される可能性があるため、これらの機能については、以下のセキュリティ制限がかけられている。

（1）　携帯電話のネイティブ機能にアクセスする際には、必ず、ユーザにネイティブ機能へのアクセス確認を行う。確認後、ネイティブ機能へのアクセスが行われる。
（2）　携帯電話に保存されている情報は、携帯電話端末から外部に持ち出せない。携帯電話に保存されている情報は、XString という特別なクラスを通してのみ i アプリで扱うことができる。しかし、このクラスは、通常の String クラスへの代入を行うことができないフレームワークが構築されている。メール送信や、HTTP 通信、赤外線通信等の送信データとして利用可能なクラスは String クラスなので、携帯電話に保存されている情報は、送信データとして外部に持ち出すことはできない。
（3）　携帯電話に保存されている情報を扱うことができるのは、公式サイトの i アプリに限られる。携帯電話に保存されている情報を参照するアプリケーションを実行する際には、i モードセンタと通信を行い、当該アプリケーションが NTT ドコモの認証を受けているものかどうかの確認が行われる。NTT ドコモの認証を受けたアプリケーションをいわゆる i アプリ DX と呼び、i アプリ DX を登録できるのは

現状では公式サイトのみである。i モードメニューに掲載されている公式サイト以外の非公式サイトでは i アプリ DX を配信できない。

表 21　XString クラス概要

XString 携帯電話のネイティブが管理しているデータで、携帯電話外に持ち出せない文字列データを表す XObject を定義する
java.lang.Object com.nttdocomo.lang.XObject com.nttdocomo.lang.XString
public int length() XString の文字列の長さを取得する。

表 22　XObject クラス概要

XObject 携帯電話のネイティブが管理しているデータで、携帯電話外に持ち出せないデータを表すオブジェクトの基底クラス
java.lang.Object com.nttdocomo.lang.XObject
public final boolean equals(java.lang.Object obj) インスタンスの同一性をチェックする。
public final int hashCode() このオブジェクトのハッシュ値を取得する。
public final String toString() オブジェクトの文字列表現を取得する。

以上のように、最新の DoJa プロファイルでは、ネイティブ機能との連携や、携帯電話端末で管理されているプライベートなデータなどを参照することは可能である。ただし、そのような Java アプリケーションは公式サイトからのみダウンロード可能なトラステッド i アプリとしてしか実装できず、非公式サイトから配布できる i アプリはセキュリティが保護された API しか利用できない。また、トラステッド i アプリといえども、セキュリティ上問題がある API の実行時には必ずユーザに実行の確認を取る仕組みになっており、また、携帯電話端末のデータは、XObject の実装により外部には持ち出せなくなっている。

3.3.2 MIDP ライブラリ

JCP により策定された MIDP の仕様には MIDP 1.0 と MIDP 2.0 の 2 つのバージョンがある。現在、MIDP は KDDI(au)や vodafone の携帯電話端末で採用されており、実装されているのは MIDP 1.0 仕様であるが、今後は MIDP 2.0 仕様へ移行していくものと思われる。

表 23　MIDP 仕様

仕様	JSR 番号	内容
MIDP-1.0	JSR37	Mobile Information Device Profile for the J2ME Platform
MIDP-2.0	JSR118	Mobile Information Device Profile2.0

MIDP 仕様では下記のパッケージ群が提供されている。MIDP 2.0 においては、ユーザインタフェースライブラリやメディア関連、ゲーム関連のライブラリが拡張されただけでなく、セキュリティ機能が大きく追加策定されている。以下、MIDP プロファイルとして策定されている API の中で通信関連やネイティブリソースアクセス関連、セキュリティ関連の API に着目し、その機能および実装を検証する。

表 24　MIDP パッケージ

パッケージ名	MIDP 1.0	MIDP 2.0	概要
javax.microedition.lcdui	○	○	ユーザインタフェース
javax.microedition.lcdui.game	×	○	ゲーム
javax.microedition.midlet	○	○	MIDlet
javax.microedition.io	○	○	ネットワーク通信
javax.microedition.pki	×	○	公開鍵
javax.microedition.media	×	○	モバイルメディア
javax.microedition.media.control	×	○	プレイヤ制御
javax.microedition.rms	○	○	格納データアクセス

（1）MIDP 1.0

MIDP 1.0 は、CLDC コンフィグレーションに採用された機器を前提に考えられたプロファイルである。MIDP が対象としているのは、小さな画面、テンキーとそれを補足する 4～5 個のキーのみを持っている機器であり、基本的には携帯電話端末を前提としている。MIDP 1.0 には、主に携帯電話端末の Java 環境のユーザインタフェースなど、アプリケーションレベル向けに策定された仕様である。

MIDP 1.0 の通信関係の API は、必要最低限のプロトコルのみ定義されている。基本的な HTTP プロトコルは定義されているが、HTTPS のようなセキュアなプロトコルは定義されていない。このため、後で述べる KDDI(au)や vodafone では、MIDP 1.0 仕様を採用してはいるものの、HTTPS などの不足しているセキュリティ機能については、独自の拡張プロファイルにて確保している。

表 25　HttpConnection インタフェース概要

HttpConnection 　　HTTP コネクションのインタフェース
interface javax.microedition.io.Connection 　　interface javax.microedition.io.InputConnection 　　interface javax.microedition.io.OutputConnection 　　　　interface javax.microedition.io.StreamConnection 　　　　　　interface javax.microedition.io.CommConnection 　　　　　　interface javax.microedition.io.ContentConnection 　　　　　　　　interface javax.microedition.io.HttpConnection
public String getURL() 　　コネクション URL 表現を返す。
public String getProtocol() 　　http や https などの URL のプロトコル名を返す。
public String getHost() 　　URL のホスト情報を返す。
public String getFile() 　　URL のファイル名の部分を返す。
public String getRef() 　　URL の#文字以降の参照文字列を返す。
public String getQuery() 　　URL の?文字以降のクエリ部分を返す。
public int getPort() 　　URL のポート番号を返す。
public String getRequestMethod() 　　HEAD、GET、POST などの現在のリクエスト方式を返す。
public void setRequestMethod(String method () 　　リクエスト方式を設定する。
public String getRequestProperty(String key) 　　リクエスト方式を設定する。
public void setRequestMethod(String method () 　　リクエストのプロパティ値を取得する。
public void setRequestProperty(String key, String value) 　　リクエストのプロパティ値を設定する。
public int getResponseCode() 　　レスポンスのコードが返される。
public String getResponseMessage() 　　レスポンスのメッセージが返される。
public int getPort() 　　HTTP ヘッダの expires フィールドの値を返す。
public long getDate() 　　HTTP ヘッダの date フィールドの値を返す。
public long getLastModified() 　　HTTP ヘッダの last-modified フィールドの値を返す。
public String getHeaderField(String name) 　　HTTP ヘッダの named フィールドの値を返す。
public int getHeaderFieldInt(String name, int def) 　　HTTP ヘッダの指定フィールドの値を数字にして返す。
public long getHeaderFieldDate(String name, long def) 　　HTTP ヘッダの指定フィールドの値を日付にして返す。
public String getHeaderField(int n) 　　HTTP ヘッダのインデックスで指定したフィールドの値を返す。
public String getHeaderFieldKey(int n) 　　HTTP ヘッダのインデックスで指定したフィールド名を返す。

MIDP 1.0 ではさらに下記の 3 つのパッケージが利用可能となっている。

- javax.microedtion.lcdui

  ユーザインタフェースを実現するためのクラスを提供するパッケージである。これはインタフェース関連であるため、セキュリティの上で問題となるクラス等の API は含まれていない。

- javax.microediton.midlet

  MIDP を用いたアプリケーション(MIDlet)を作成する際に継承しなくてはならないクラスが提供された MIDP 固有のパッケージである。このパッケージの MIDlet と MIDletStateChangeException には、ネイティブなリソースにアクセスするようなメソッドは定義されていない。

- javax.microedition.rms

  永続的に保存可能なデータをデータストレージに記憶させ、それを読み込む機能を実現するクラスを提供するパッケージである。データストレージに記憶されたデータは、複数の MIDlet をまとめた MIDlet スイートと呼ばれる同じ MIDlet スイート内の MIDlet 間でしか共有できず、他の MIDlet スイートの MIDlet のデータストレージをアクセスすることはできない。

MIDP 1.0 コアとなるパッケージは CLDC 1.0 と同じであり、java.io パッケージや java.lang パッケージをそのまま利用している。そのため、その安全性は変わらない。また、HTTP 通信機能は規定されてはいるが､セキュリティ上問題のある API は付加されていない。

（2）MIDP 2.0

MIDP 2.0 における MIDP 1.0 からの主なセキュリティ上の拡張は以下の通りである。

・ ネットワーク接続性の拡張
・ プッシュ・アーキテクチャのサポート
・ ドメインセキュリティモデル、アプリケーションへの署名および証明書の検証

- ネットワーク接続性の拡張

MIDP 2.0 では、HTTPS、データグラム、ソケット、サーバーソケットおよびシリアルポート接続にも対応がなされている。CLDC の GCF(Generic Common Framework)を利用するようになっているため、MIDP レベルで用意される API は、通信のコネクションを作成するためにパラメータを設定したり、属性値を参照したりするものでしかない。

以下の表 26から、表 32までの各インタフェースは、MIDP 2.0 仕様で新たに策定されたプロトコル、セキュリティ情報に関連するインタフェースの概要である。

表 26　CommConnection インタフェース概要

CommConnection シリアルポートコネクションのインタフェース
interface javax.microedition.io.Connection interface javax.microedition.io.InputConnection interface javax.microedition.io.OutputConnection interface javax.microedition.io.StreamConnection interface javax.microedition.io.CommConnection
public int getBaudRate() シリアルポートコネクションのボーレートを取得する。
public int setBaudRate(int baudrate) シリアルポートコネクションのボーレートを設定する。

表 27　HttpsConnection インタフェース概要

HttpsConnection HTTPS コネクションのインタフェース
interface javax.microedition.io.Connection interface javax.microedition.io.InputConnection interface javax.microedition.io.OutputConnection interface javax.microedition.io.StreamConnection interface javax.microedition.io.CommConnection interface javax.microedition.io.ContentConnection interface javax.microedition.io.HttpConnection interface javax.microedition.io.HttpConnection
public SecurityInfo getSecurityInfo() サーバにつながったコネクションに関連するセキュリティ情報を返す。
public int getPort() HTTPS コンネクションのポート番号を返す。

表 28　SecureConnection インタフェース概要

SecureConnection セキュアコネクションのインタフェース
interface javax.microedition.io.Connection interface javax.microedition.io.InputConnection interface javax.microedition.io.OutputConnection interface javax.microedition.io.StreamConnection interface javax.microedition.io.SocketConnection interface javax.microedition.io.SecureConnection
public SecurityInfo getSecurityInfo() サーバにつながったコネクションに関連するセキュリティ情報を返す。

表 29　SecurityInfo インタフェース概要

SecurityInfo 　　セキュアコネクション情報のインタフェース
interface javax.microedition.io.SecurityInfo
public Certificate getServerCertificate() 　　サーバとのコネクションに使われた証明書を返す。
public String getProtocolVersion() 　　プロトコルのバージョン番号を返す。
public String getProtocolName() 　　セキュアなプロトコル名を返す。
public String getCipherSuite() 　　コネクションに利用した暗号スイートを返す。

表 30　ServerSocketConnection インタフェース概要

ServerSocketConnection 　　サーバソケットストリームコネクションのインタフェース
interface javax.microedition.io.Connection 　　interface javax.microedition.io.StreamConnectionNotifier 　　　　interface javax.microedition.io.ServerSocketConnection
public String getLocalAddress() 　　バインドされたソケットのローカルアドレスを返す。
public int getLocalPort() 　　バインドされたソケットのローカルポートを返す。

表 31　SocketConnection インタフェース概要

SocketConnection 　　ソケットストリームコネクションのインタフェース
interface javax.microedition.io.Connection 　　interface javax.microedition.io.InputConnection 　　interface javax.microedition.io.OutputConnection 　　　　interface javax.microedition.io.StreamConnection 　　　　　　interface javax.microedition.io.SocketConnection
public void setSocketOption(byte option, int value) 　　コネクションのソケットオプションを設定する。
public int getSocketOption(byte option) 　　コネクションのソケットオプションを取得する。
public String getLocalAddress() 　　バインドされたソケットのローカルアドレスを取得する。
public int getLocalPort() 　　バインドされたソケットのローカルポートを取得する。
public String getAddress() 　　バインドされたソケットのリモートアドレスを取得する。
public int getPort() 　　バインドされたソケットのリモートーポートを取得する。

表 32　UDPDatagramConnection インタフェース概要

UDPDatagramConnection 　　データグラムコネクションのインタフェース
interface javax.microedition.io.Connection 　　interface javax.microedition.io.DatagramConnection 　　　interface javax.microedition.io.UDPDatagramConnection
public String getLocalAddress() 　　バインドされたソケットのローカルアドレスを取得する。
public int getLocalPort() 　　バインドされたソケットのローカルポートを取得する。

- プッシュ・アーキテクチャのサポート

サーバからの情報のプッシュ配信がサポートされ、サーバとアプリケーションが連携できるようになった。MIDlet を登録しておくと、携帯電話端末がサーバから情報を受け取った時点で MIDlet をアクティブ化できる。アプリケーション管理ソフトウェアの実装に依存するが、PushRegistry クラスは、ドメインプロテクションのセキュリティフレームワークとパーミッションにより保護される。

表 33　PushRegistry クラス概要

PushRegistry 　　着信接続のリストを維持するクラス
java.lang.Object 　　javax.microedition.io.PushRegistry
public static void registerConnection(String connection, String midlet, String filte) 　　アプリケーション管理ソフトウェアに動的コネクションを登録する。
public static boolean unregisterConnection(String connection () 　　動的コネクションを削除する。
public static String[] listConnections(boolean available) 　　MIDlet スイートに登録されているコネクションの一覧を返す。
public static String getFilter(String connection) 　　要求されたコネクションに登録されたフィルタを取得する。
public static long registerAlarm(String midlet, long time) 　　アプリケーションを実行する時刻を登録する。

- ドメインセキュリティモデル、アプリケーションへの署名および証明書の検証

HTTPS の他にもエンド・ツー・エンドレベルのセキュリティが MIDP 仕様において規定されている。Java アプリケーションの発行者が、作成した Java アプリケーションにディジタル署名する。公開鍵方式を利用することで、配布アプリケーションの完全性と出所を確認できる機能である。Java アプリケーションのダウンロード元が国内通信キャリアか、一般のコンテンツプロバイダであるかを区別し、アドレス帳へのアクセスを許可するかどうかなど、アプリケーションによってどの程度までの機能を利用できるかに差をつけるドメインセキュリティモデルや、SSL による安全な通信機能など、セキュリティ機能が強化されている。

ドメインセキュリティモデルは、セキュリティレベルに応じてアプリケーションが利用できる機能を制限できる仕組みであり、アプリケーションに署名や証明書がないと、電話帳へアクセスできないなどの仕組みが構築できる。

表 34　Certificate のインタフェース概要

Certificate 　　証明書のインタフェース
interface javax.microedition.pki.Certificate
public String getSubject() 　　証明書の主体者名を取得する。
public String getIssuer() 　　証明書の発行者名を取得する。X.509 の証明書であれば、"X.509"を返す。
public String getType() 　　証明書のタイプを取得する。
public String getVersion() 　　証明書のバージョン番号を取得する。X.509 の証明書であれば、"2"を返す。
public String getSigAlgName() 　　証明書の署名アルゴリズムを取得する。
public long getNotBefore() 　　証明書の有効期間の開始時刻を取得する。
public long getNotAfter() 　　証明書の有効期間の終了時刻を取得する。
public String getSerialNumber() 　　証明書のシリアル番号を取得する。

MIDP 2.0 は、MIDP 1.0 によって与えられているセキュリティモデルにアプリケーションレベルのセキュリティ、およびエンド・ツー・エンドレベルのセキュリティモデルが付加されたものである。様々な通信プロトコル以外にもセキュアな通信プロトコルが規定され、各メソッドも規定されている。

データの読み書きに必要なレコードの定義と、その取り扱い（比較やフィルタリング）が可能な API が規定されているが、1 つの Java アプリケーションにつき 1 つのデータストレージが割り当てられ、他のアプリケーションとは共有できないようになっている。したがって、他のアプリケーションのデータを破壊したり、他のアプリケーションが管理しているプライベートなデータなどを参照したりすることはできない。

しかしその一方で、MIDP 2.0 仕様では携帯電話端末のネイティブデータにアクセスするようなライブラリも規定されている。そこでプロテクションドメインを利用して、任意のレベルで API を保護する仕組みが提供されている。

3.3.3　KDDI-P ライブラリ

KDDI-P は、KDDI(au)独自のサービスを KDDI(au)の Java 環境で利用するためのプロファイルであり、KDDI(au)の携帯電話端末に特化された機能を利用するための拡張 API が提供されている。

KDDI-P は EZ アプリの開発用に公開されているライブラリであり、CLDC ライブラリと MIDP ライブラリの上位に位置した携帯電話端末機能を拡張するものである。J2ME 標準のライブラリと合わせて利用することより KDDI(au)の携帯電話端末固有の機能操作や携帯電話端末内の情報参照などが可能になる。

表 35　KDDI-P のパッケージ概要

KDDI-P パッケージ名	Phase1	Phase2	Phase2.5	概要
com.jblend.media	×	×	○	メディアプレイヤ
com.jblend.media.smaf	×	×	○	SMAF プレイヤ
com.jblend.media.smaf.phrase	×	×	○	フレーズ
com.jblend.net	×	○	○	ユーティリティ
com.kddi.io	○	○	○	入出力
com.kddi.matiuke	×	×	○	待ち受け設定
com.kddi.media	○	○	○	メディアプレイヤ
com.kddi.system	○	○	○	端末情報アクセス

KDDI(au)の Java 環境のバージョンは「Phase（フェーズ）」で表されており、機種にも依存するが、以下のように段階的にバージョンアップされている。各 Phase において携帯電話端末のリソースにアクセスできる機能が実装されている。

表 36　KDDI-P 環境における各 Phase の主な実装機能

項番	主な機能	Phase1	Phase2	Phase2.5
［1］	アドレス帳の参照	○	○	○
［2］	位置情報の取得	○	○	○
［3］	EZ アプリ通信（CMAIL）の制御	○	○	○
［4］	データフォルダの制御	○	○	○
［5］	HTTP 通信	×	○	○
［6］	ブラウザ連携（URL-to）	×	○	○
［7］	メーラ連携（Mail-to）	×	○	○
［8］	電話連携（Phone-to）	×	○	○
［9］	待ち受け設定	×	×	○

［1］アドレス帳の参照

端末に搭載されている電話帳を起動し、電話帳終了時に選択した 1 レコードの電話番号等を取得できる。端末に搭載されている電話帳が起動している間、EZ アプリはブロッキングされていて、電話帳機能が終了するのを待つ。電話帳検索画面が起動されることにより API が実行されていることが判明する。しかし、アドレス帳にデータを登録することはできない。

表 37　AddressBook

AddressBook 　　アドレス帳データを参照するためのクラス
java.lang.Object 　　com.kddi.system.AddressBook
public static PersonalInfo getEmailAddress() 　　移動機の電話帳検索画面を呼び出し、利用者が選択した登録データの電子メールアドレスを呼び出し側に返す。
public static PersonalInfo getTelNo() 　　移動機の電話帳検索画面を呼び出し、利用者が選択した登録データの電話番号を呼び出し側に返す。

PersonalInfo は個人情報を保持するためのクラスであり、取得できる情報は端末によって異なるが、サポートしていない情報または設定されていない情報を取得した場合は null が返される。PersonalInfo クラスは、AddressBook クラスと合わせて利用する。

表 38　PersonalInfo

PersonalInfo 　　個人情報をカプセル化したクラス
java.lang.Object 　　com.kddi.system. PersonalInfo
public String getTelNo() 　　電話番号を取得する。
public String getEmail() 　　電子メールアドレスを取得する。

［2］位置情報の取得

「基地局方式」で測位した現在位置情報を取得することができる。位置情報をアプリケーションが取得する際には、原則的として携帯電話端末ユーザの意思確認を必要とする。ただし、ユーザが「意思確認を必要としない設定」をしていた場合は、確認画面が表示されない。

表 39　Location

Location 　　基地局方式による位置情報取得のためのクラス
java.lang.Object 　　com.kddi.system.Location
public static Location getLocation() 　　システムで唯一の位置情報インスタンスを返却する。ユーザが位置情報利用を許可しなかったときは null を返却する。
public String getLat() 　　現在地の緯度情報が返却される。取得した値の単位は、getDatum, getUnit によって定まる。
public String getLon() 　　現在地の経度情報が返却される。取得した値の単位は、getDatum, getUnit によって定まる。
public String getDatum() 　　現在地を取得した時の測地系が返却される。
public String getUnit() 　　現在地を取得した時の座標系が返却される。

［3］EZ アプリ通信（CMAIL）の制御

パケット C メールを利用した EZ アプリ通信による EZ アプリ同士の通信機能が利用できる。EZ アプリ通信には「単発モード」と「連続モード」があり、「連続モード」は KDDI(au) によって公式の認定を受けた場合に利用できる。

C メール機能において非通知設定されている場合、EZ アプリからの EZ アプリ通信発信もできない。

「単発モード」では、発信側の EZ アプリが着信側の EZ アプリの起動を要求し、着信側で、起動要求された EZ アプリが存在すれば起動する。起動された EZ アプリは、発信側の電話番号やニックネーム、発信側 EZ アプリからのメッセージを取得できる。

表 40　CMailConnection

CMailConnection 　　C メールによる通信を制御するのためのインタフェース
interface javax.microedition.io.Connection 　　interface com.kddi.io.BrowserConnection 　　interface javax.microedition.io.DatagramConnection
public void setSingleMode() 　　送信モードを単発メールモードに設定する。
public void setMultipleMode() 　　送信モードを連続メールモードに設定する。
public void setNickName(String nickname) 　　ニックネームを設定する。未設定の場合には空文字が設定される。
public void setMessage(String message) 　　送信メッセージを設定する。
public String getTelNo() 　　現在地を取得した時の測地系が返却される。
public String getDatum() 　　通信相手の電話番号を取得する。
public String getNickName() 　　通信相手のニックネームを取得する。
public String getMessage() 　　通信相手からのメッセージを取得する。
public int getMode() 　　通信モードを取得する。
public int accept() 　　連続モードでの接続要求を受けている状態で、受諾要求を発行する。
public int connect() 　　連続モードで接続要求を発行する。
public void sendTo() 　　単発メールを送信する。

［4］データフォルダの制御

データフォルダは階層的フォルダ構造を有するファイルシステムで、データフォルダに存在する全てのフォルダに含まれている全てのファイルを階層構造に関係なく取得できる。リストのエントリは、下記の情報を提供する。

（1） データファイルのパス名
（2） データファイルのサイズ（64 ビットの符号無し整数値）
（3） タイプ（「disposition」データタイプ ）
（4） 著作権情報（コピープロテクトされているか否か）
（5） ユーザに対する表示名

EZ アプリケーションはこれらの情報の取得が可能だが、データファイルの中身の読み込みはできず、ファイルへの書き込みも許容されていない。

表 41 DataFolderConnection

DataFolderConnection データフォルダにアクセスするための Connection インタフェース
interface javax.microedition.io.Connection interface javax.microedition.io.InputConnection interface javax.microedition.io.OutputConnection interface javax.microedition.io.StreamConnection interface javax.microedition.io.ContentConnection interface com.kddi.io.DataFolderConnection
public String[] getList() フォルダ以外の全ファイルリストを取得する。
public String[] getList(int type) 指定したタイプに応じたファイルリストを取得する。
public boolean isCopyrighted() 著作権フラグが設定されているかどうか判別する。
public String getName() ファイルの表示名を取得する。
public String getType() ファイルのタイプを取得する。

［5］HTTP 通信

2001 年 7 月に発売された端末（Phase1 仕様）では EZ アプリから HTTP/HTTPS 通信はできなかったが、2001 年 12 月以降の端末（Phase 2 仕様）からは HTTP/HTTPS による通信も可能になっている。

携帯電話端末機能による EZ アプリの通信動作制限も可能であり、ユーザは HTTP 通信を利用する EZ アプリ起動時に確認画面を出すか出さないか、または一切 HTTP 通信を利用しないかの設定を行うことができる。ユーザは通信確認設定画面において以下の 3 通りの選択ができる。

表 42　通信確認設定

通信確認設定	内容
ON（確認なし）	HTTP 通信を利用する Java アプリを起動した時に、通信実行確認画面を出さない。
ON（確認あり）	HTTP 通信を利用する Java アプリを起動した時に、通信実行確認画面を出して、ユーザの意志確認を行う。ユーザの選択に応じて HTTP 通信機能を利用する
OFF	Java アプリからは一切 HTTP 通信を利用しない。

［6］ブラウザ連携（URL-to）

EZ アプリから指定の URL へブラウザで接続したり、ブラウザから指定の EZ アプリを起動したりすることができる。ただし、EZ アプリからブラウザを起動した場合は、ブラウザが起動した時点で、実行中の EZ アプリは終了する。

BrowserConnection は、Connector.open メソッドの "url:"、"urls:" スキームに対応し、ブラウザを起動するために使用する。"url:〜" という URL 文字列の先頭部分を "http:〜" に変換してブラウザを起動する。同様に、"urls:〜" の場合には "https:〜" に変換する。

表 43　BrowserConnection

BrowserConnection URL-to 機能をサポートするための Connection インタフェース
interface javax.microedition.io.Connection interface com.kddi.io.BrowserConnection

［7］メーラ連携（Mail-to）

EZ アプリから、「TO」「CC」「BCC」「題名」「本文」を指定して、メールを送信することができる。添付ファイルは EZ アプリから指定できず、メールの送信が完了すると EZ アプリに戻る。

IMAP4MailConnection は、Mail-to 機能を利用してメーラを起動するために使用する。MIDP の Connector.open スタティックメソッドに "mailto:" スキームの URL を指定すると、IMAP4MailConnection インタフェースを実装したオブジェクトが取得できる。Mail-to 機能を使用するためには、このオブジェクトのメソッドを使用する。

表 44　IMAP4MailConnection

IMAP4MailConnection 　　IMAP4 メールを扱うための Connection インタフェース
interface javax.microedition.io.Connection 　　interface com.kddi.io.IMAP4MailConnection
public void addTo(String address) 　　To:を追加する。追加できるアドレスの長さは 64 バイト以内。
public void addTo(PersonalInfo person) 　　To:を追加する。
public String[] getTo() 　　送信先リストを取得する。
public void addCc(String address) 　　Cc:を追加する。追加できるアドレスの長さは 64 バイト以内。
public void addCc(PersonalInfo person) 　　Cc:を追加する。
public String[] getCc() 　　カーボンコピー送信先リストを取得する。
public void addBcc(String address) 　　Bcc:を追加する。追加できるアドレスの長さは 64 バイト以内。
public void addBcc(PersonalInfo person) 　　Bcc:を追加する。
public String[] getBcc() 　　ブラインドカーボンコピー送信先リストを取得する。
public void setContent(String content) 　　本文を設定する。設定できる本文の長さは 1000 バイト以内。
public String getContent() 　　本文を取得する。
public void setSubject(String subject) 　　件名を設定する。設定できる件名の長さは 100 バイト以内。
public String getSubject() 　　件名を取得する。
public void send() 　　メールを送信する。
public long getLength() 　　メール本文の長さを取得する。

［8］電話連携（Phone-to）

EZ アプリから電話番号を指定して、音声通話を発信することができる。通話の間 EZ アプリはサスペンドされ、通話が完了するとリジュームにより EZ アプリに戻る。

PhoneConnection クラスは、Phone-to 機能を利用して電話をかけるために使用する。MIDP の Connector.open スタティックメソッドに "phoneto:" スキームの URL を指定すると、PhoneConnection インタフェースを実装したオブジェクトが得られる。Phone-to 機能を使用するためには、このオブジェクトのメソッドを使用する。

表 45　PhoneConnection

PhoneConnection Phone-to 機能をサポートするためのインタフェース
interface javax.microedition.io.Connection interface com.kddi.io.PhoneConnection

［9］待ち受け設定

Phase2.5 対応端末からは全ての EZ アプリに対する待ち受け設定が可能である。また、アラームの発生、電話の着信、C メールの着信、および E メールの着信イベントを利用した EZ アプリを作成することができる。com.kddi.matiuke パッケージの配下にあるクラスやインタフェースを利用する。

以上のように、KDDI-P を利用することで、ネイティブ機能との連携や、携帯電話端末で管理されているデータなどを参照することは可能である。ただし、そのような Java アプリケーションは公式サイトからしか利用できない運用になっている。非公式サイトから配布できる EZ アプリは、セキュリティが保護された API しか利用できない。また、KDDI-P は CLDC を拡張するクラスライブラリであるため、ファイルの入出力に関する機能が実装されておらず、com.kddi.io パッケージにおいてもデータフォルダのエントリを参照するための DataFolderConnection インタフェースがあるのみであった。

公式サイトから利用可能な EZ アプリにおいても、電話帳を参照するようなセキュリティ上問題がある API の実行時には、必ずユーザに実行の確認を取る仕組みになっている。

3.3.4　JSCL ライブラリ

JSCL(J-PHONE Specific Class Library)は、vodafone 独自のサービスを vodafone の携帯電話の Java 環境で利用するためのクラスライブラリのセットである。

JSCL はＶアプリの開発用に公開されているライブラリであり、CLDC ライブラリと MIDP ライブラリの上位に位置した携帯電話端末機能を拡張するプロファイルである。CLDC や MIDP の J2ME 標準のライブラリと合わせて利用することより vodafone の携帯電話端末固有の操作や情報取得が可能になる。

V アプリの拡張 API の仕様には JSCL 1.0(C4 型)、JSCL 1.1(P4 型)、JSCL 1.2(P5 型)がある。機種に応じて実装されているライブラリが異なる。

表 46　JSCL パッケージ概要

JSCL パッケージ名	JSCL 1.0	JSCL 1.1	JSCL 1.2	概要
com.j_phone.amuse	○	×	×	フレーズプレイヤ
com.j_phone.amuse.j3d	○	×	×	3D
com.j_phone.io	×	×	○	入出力
com.j_phone.media	○	×	×	メディアプレイヤ
com.j_phone.midlet	×	○	○	MIDlet
com.j_phone.phonedata	×	×	○	ネイティブ連携
com.j_phone.system	○	○	○	固有デバイス制御
com.j_phone.ui	×	×	○	ユーザインタフェース
com.j_phone.util	○	○	○	ユーティリティ
com.jblend.graphics.j3d	×	○	○	3D ポリゴン機能
com.jblend.graphics.sprite	×	○	○	スプライト機能
com.jblend.io deflate	×	○	○	deflate 形式
com.jblend.media	×	○	○	メディアプレイヤ
com.jblend.media.jpeg	×	○	○	JPEG 形式
com.jblend.media.mng	×	○	○	MNG 形式
com.jblend.media.png	×	○	○	PNG 形式
com.jblend.media.smaf	×	○	○	SMAF 形式
com.jblend.media.smaf.phrase	×	○	○	フレーズプレイヤ
com.jblend.media.smd	×	○	○	SMD 形式
com.jblend.micro.io	×	×	○	シリアル・ポート
com.jblend.micro.lcdui	○	○	○	日本語対応

JSCL 1.2 では、com.j_phone.io パッケージや com.j_phone.phonedata パッケージ、com.j_phone.system パッケージが拡張され、V アプリから携帯電話端末のリソースにアクセスできる以下の機能が利用できるようになっている。

- ネイティブシステムとの連携
- 固有デバイスの制御
- メールの送受信
- アドレス帳の参照
- ブラウザ連携、メーラ連携、電話連携

しかし、このような API がすべて公開されているとはいえ、運用規定上、V アプリは非公式・非公認ないわゆる勝手サイトを立てることは許されていない。したがって、現状では非公式・非公認サイトから V アプリを携帯電話端末にダウンロードして利用することはできない。V アプリは、セキュリティ面を考慮して vodafone の公式サイト、または公認の配布サイトからしかダウンロードできない制限がかけられている。いかなる V アプリも、コンテンツアグリゲータと呼ばれる機関に検査を依頼しなければならない。そして、検査に合格した場合にコンテンツアグリゲータより V アプリが公認配布サイトから公開される。コンテンツアグリゲータの検査を偶然すり抜けるようなことさえなければ、不正な V アプリが配布される

ことはない。

また、コンテンツアグリゲータから配布されているVアプリに関しても、一般にネイティブな機能と連携する場合は、ダイアログがポップアップしてユーザに確認を要請するといった設計がなされている。

実行時にユーザの確認が必要となるメソッドを以下に示す。確認ダイアログにおいてユーザが「承諾」を選択した場合にのみメソッドの処理を実行し、ユーザが「キャンセル」を選択した場合はSecurityException例外を発生する。

表 47　確認ダイアログ表示メソッド一覧

クラス名	メソッド
PhoneData (com.j_phone.phonedata)	public void createElement(DataElement element) 電話関連情報リスト内に新しい要素を登録する。
	public void delete(DataElement element) 電話関連情報リスト内の要素を削除。
MailAgent (com.j_phone.system)	public void send(MailData data) メールを送信する。
	public void receiveRemainder(MailData data) メールサーバから受信していないメールの続きを受信する。
ClientObexConnection (com.j_phone.io)	public void connect() 赤外線通信の接続を確立する。
ServerObexConnection (com.j_phone.io)	public void accept() OBEXクライアントからの接続要求を待つ。
StorageConnection (com.j_phone.io)	public boolean createFolder() フォルダを新規に作成する。
	public boolean delete() ファイルまたはフォルダを削除する。
	public java.io.OutputStream openOutputStream() ファイルに対する出力ストリームを開く。
	public boolean renameTo(String newName) ファイルまたはフォルダの名前を変更する。

また、アクセスの制限されたVアプリからの使用を制限するメソッドを示す。 以下のメソッドは、アクセスの制限された V アプリから呼び出された場合にセキュリティ違反としてSecurityExceptionを発生する。アクセス制限は、ユーザが携帯電話端末上にてユーザ情報を利用するVアプリの実行の可否を設定することができるようになっている。

表 48　セキュリティ対象メソッド一覧

クラス名	メソッド
DeviceControl (com.j_phone.system)	public int getLatitude() 現在の緯度を取得する。
	public int getLongitude() 現在の経度を取得する。
	public String getPlaceName() 現在位置の地名を取得する。
	public static void setMailListener(MailListener listener) メール着信を待ち受ける MailListener を登録する。
	public static void setTelephonyListener(TelephonyListener listener) 電話の着信イベントを待ち受ける TelephonyListener を登録する。
MailAgent (com.j_phone.system)	public void send(MailData data) メールを送信する。
PhoneDataConnector (com.j_phone.phonedata)	public static PhoneData openPhoneData(String name, int index) 電話関連情報リスト（受信メールボックスリスト、送信メールボックスリスト、またはアドレス帳リスト）を取得する。
StorageConnection (com.j_phone.io)	public java.io.OutputStream openOutputStream() データフォルダへのデータファイルを保存、および作成する。
	public boolean createFolder() データフォルダにフォルダを新規に作成する。
	public java.io.InputStream openInputStream() データフォルダのファイルに対する入力ストリームを開く。
	public boolean delete() データフォルダのファイルまたはフォルダを削除する。
	public boolean renameTo(String newName) データフォルダのファイルまたはフォルダの名前を変更する。

同様に、待ち受けアプリの操作を実装する com.j_phone.midlet.ResidentMIDlet はセキュリティ対象クラスであり、以下のメソッドもアクセスの制限された V アプリから呼び出された場合にセキュリティ違反として SecurityException を発生する。

表 49　ResidentMIDlet

ResidentMIDlet 待ち受けアプリのための基底インタフェース
java.lang.Object javax.microedition.midlet.MIDlet com.j_phone.midlet.ResidentMIDlet
public abstract void ring(String name, String number) 電話がかかってきた場合に呼び出される。
public abstract void ignored() 電話が切れた場合に呼び出される。
public abstract void received(String name, String address, int detail) SMS を着信した場合に呼び出される。
public abstract void notice(String comment) アラーム時刻になった時に呼び出される。
public abstract void ringStarted() 着信通知を開始した場合に呼び出される。
public abstract void ringStopped() 着信通知を終了した場合に呼び出される。

その他、vodafone では、Java アプリケーション、メモリダイヤル、着信メロディや画像などのデータを SD カードに保存して、機種変更時に簡単にデータを移行できる仕組みが導入されている。これに対応した端末は、Java アプリケーションを SD カードから呼び出せるように改良されている。ただし、著作権やセキュリティを考慮して、ユーザがパソコンを使って SD カードにコピーした Java アプリケーションを起動することはできない。あくまでも vodafone からダウンロードした公式アプリ、または公認配布サイトからダウンロードしたオープンアプリのみ起動できるようになっている。

3.3.5 情報資産の保護

これまで見てきたように、各プロファイルについては、それぞれ仕様は異なるが、セキュリティは保護されていることが確認できた。

MIDP 1.0 は、CLDC の上位に位置しているため低レベルでのセキュリティモデルを拡張したり付加したりするものではなく、アプリケーションレベルのセキュリティについてもセキュリティ上問題がある API は付加されてはいない。また、Java アプリケーションが利用するデータストレージは、あくまで対象とする Java アプリケーションからのみアクセス可能となっており、他の Java アプリケーションが管理しているプライベートなデータなどを参照したり、データを破壊したりすることはできなくなっている。

MIDP 2.0 は、セキュアな通信プロトコルが規定され、MIDP 1.0 より安全に通信が可能となっており、データの読み書きについては、MIDP 1.0 と同様に 1 つの Java アプリケーションにつき 1 つのデータストレージが割り当てられ、他のアプリケーションとは共有できないようになっている。

しかし、MIDP 2.0 仕様では、携帯電話端末のネイティブデータにアクセスするようなライブラリも規定されている。これまでは、そのようなセキュリティ上の問題を含んでいるような API は実装しないというのが一般的であったが、プロテクションドメインを利用して、任意のレベルで API を保護する仕組みが提供されている。API 自身の保護、および実行制限を実装レベルで規定し Java のアプリケーションのセキュリティを保護するのではなく、セキュリティポリシーにより Java アプリケーションのセキュリティを保護することができる。

MIDP 1.0 プロファイルにおけるセキュリティモデルは、CLDC 1.0 コンフィグレーションのセキュリティモデルと同等であり、安全であるといえる。また、MIDP 2.0 プロファイルにおけるセキュリティモデルは、MIDP 1.0 プロファイルのセキュリティモデルをより柔軟に拡張したセキュリティモデルとなっており、適切に運用すれば十分に安全であるということができる。

最新の DoJa プロファイルでは、ネイティブ機能との連携や、携帯電話端末で管理されているプライベートなデータなどを参照することが可能である。ただし、そのような Java アプリケーションは「トラステッド i アプリ」として公式サイトからのみダウンロードを可能とし、非公式サイトから配布できる i アプリは、セキュリティが保護された API しか利用でき

ない。また、トラステッド i アプリといえども、セキュリティ上問題がある API の実行時には必ずユーザに実行の確認を取る仕組みになっており、さらに、携帯電話端末のデータは XObject の実装により外部には持ち出せなくなっている。

DoJa プロファイルのセキュリティは、運用と組み合わせることで保護されている。

KDDI-P でも、ネイティブ機能との連携や、携帯電話端末で管理されているデータなどを参照することが可能である。ただし、そのような Java アプリケーションは公式サイトからしか利用できない運用になっており、非公式サイトから配布できる EZ アプリはセキュリティが保護された API しか利用できない。また、公式サイトから利用可能な EZ アプリにおいても、電話帳を参照するようなセキュリティ上問題がある API の実行時には、必ずユーザに実行の確認を取る仕組みになっている。

KDDI-P のセキュリティも、運用と組み合わせることで保護されている。

JSCL プロファイルには、個人情報のアクセスなどのネイティブな機能を利用する API などが提供されているが、すべての V アプリは検査され、vodafone が認定したコンテンツアグリゲータと呼ばれるサイト運営者のインターネットサーバからのみ公開できるという運用を取っている。

このように JSCL のセキュリティも、運用と組み合わせることで保護されている。

4 VM のセキュリティモデルとその課題

現在、国内通信キャリアが提供する Java 搭載携帯電話の JavaVM としては、主にアプリックス社の JBlend が採用されている。JBlend は、vodafone と KDDI が全端末共通で採用し、NTT ドコモでも一部の機種（SH，SO）で採用している。NTT ドコモの Java 搭載携帯電話に搭載される JavaVM については非公開となっているが、JBlend を採用していないベンダについては、Sun Microsystems 社 の KVM をもとに独自に開発しているようである。

表 50　国内通信キャリアが提供する JavaVM

	NTT ドコモ	vodafone（J-Phone）	KDDI（au）
ハードベンダ	ソニーエリクソン、NEC、松下電器、三菱電機、富士通、シャープ	シャープ、NEC、サンヨー、松下電器、東芝、三菱電機	日立、サンヨー、カシオ、ソニーエリクソン、東芝、京セラ
JavaVM	ベンダ開発、または JBlend	JBlend	JBlend

今回は、J2ME のコアとなる Sun Microsystems 社の JavaVM である KVM とアプリックス社の JBlend、海外向けの Java 搭載携帯電話に採用されている実績のある ACCESS 社の JV-Lite2 について、各 VM が実現しているセキュリティモデルを調査した。

4.1 KVM のセキュリティ調査

4.1.1 KVM の特徴

KVM は、CLDC 仕様を実現する VM の 1 つであり、J2ME 用に Sun Microsystems 社が実装したものである。

KVM は、Java 仮想マシン仕様（JVM）がもとになっており、数 100K バイトの合計メモリ容量しかなく、資源に制約のあるデバイスで動作する VM として作成された、Java 仮想マシンのサブセットである。具体的には、以下のような仕様で設計されている。

- VM 自体の占有メモリサイズは 40KB から 80KB に抑える
- クリーンで移植性が高い
- モジュール式でカスタマイズが可能
- 他の設計目標を損なうことなくできるだけ完全で高速である

KVM が要求する最小合計メモリサイズは約 128KB であり、40K～80KB の VM 自体の他、最小限の Java ライブラリやアプリケーション実行のためのヒープ領域として利用される。このようなメモリサイズの制限やセキュリティ上の理由により、JVM で定義されている機能の一部が制限されている。

KVM は、CLDC のリファレンス実装であるという位置付けから、以降では CLDC のセキュリティモデルについて調査する。

4.1.2 CLDC のセキュリティモデル

Java アプリケーションはネットワーク上で使用されることを前提に設計されており、Java 環境のセキュリティモデルでは、ローカル環境を Java プログラムから保護することを目標としている。これを実現するために導入されたのがサンドボックスモデルである。サンドボックスモデルは、「Java アプリケーションは閉じた環境内で実行されなければならない」ことを意味する。

J2SE のセキュリティモデルの実現に使用されるメモリ容量は、CLDC をサポートする VM に用意されたメモリ容量を遥かに上回るため、CLDC では J2SE とは別の方法でサンドボックスモデルを実現した。

CLDC のサンドボックスモデルでは、以下のことが要求される。

- Java クラスファイルが正しく検証され、妥当な Java アプリケーションであることが保証されている
- JavaAPI は、CLDC、プロファイル、メーカごとの拡張プロファイルで定義された JavaAPI に制限され、アプリケーションプログラマは、これらの事前に定義された API のみを利用することができる

・ Java アプリケーションのダウンロードや管理は、VM に元から組み込まれているクラスローディング機構が実行し、ユーザ定義クラスローダは提供されない。これによりプログラマが VM 標準のクラスローディング機構をオーバーライドすることを防ぐ
・ アプリケーションプログラマは、ネイティブな機能を含む新しいライブラリをダウンロードしたり、事前に定義されている Java ライブラリでないネイティブな機能にアクセスすることができない

このようなサンドボックスモデルは、以下のような仕組みによって実現されている。
・ クラスローダ
・ クラスファイルベリファイア

4.1.3 クラスローダ

クラスローダは、クラスファイルを VM にロードする（読み込む）機能である。クラスファイルのロードには VM にもともと組み込まれているクラスローディング機構（システムクラスローダ）を使う方法と、クラスを読み込む方法を java.lang.ClassLoader を使用して独自に作成したユーザ定義のクラスローダを使う方法がある。CLDC ではリソース制限があるため、セキュリティ上不完全なユーザ定義のクラスローダは提供されない。したがって、システムクラスローダがすべてのクラスファイルをロードする。また、アプリケーションプログラマは、クラスファイルの検索順序を変更したり、システムクラスをオーバーライドすることができない。

KVM には、クラスローダとして、クラスファイルをディレクトリ内のファイルや圧縮された JAR ファイル内のエントリとして読み取る実装が含まれている。また、移植の際にオプションとして CLASSPATH のサポートの有無を設定することで、CLASSPATH で指定したクラスファイルをシステムクラスローダで読み込むかどうかを指定することができる。

4.1.4 クラスファイルベリファイア

クラスファイルベリファイアは、クラスローダによってロードされたクラスファイルの安全性を検証するシステムである。

J2SE のクラスファイルベリファイアは占有するメモリ領域が大きいため、CLDC では必要メモリの少ない効率的な検証方法が検討され、KVM でも実装されている。

このクラスファイルの検証は、通常のクラスファイルベリファイアを 2 フェーズに分けて実行している。

（1）フェーズ 1：デバイス外の事前検証

これは、Java アプリケーションを開発しているコンピュータ上で行われる検証である。これにより、実行時の負荷が少なくなるようにしている。

事前検証では、プリベリファイツールが以下の 2 つの動作を実行する。

① すべてのサブルーチンをインライン展開する

jsr（サブルーチンへジャンプ）、ret（サブルーチンの呼び出し元へ制御を戻す）といったバイトコードをクラスファイルから削除して、サブルーチンをインライン展開する。

② クラスファイルに StackMap 属性を追加する

StackMap 属性には、フレームの状態を表す情報を記述している。フレームとは、メソッドが使用する作業用のメモリ領域であり、ローカル変数とオペランドスタックから構成されている。ローカル変数には、インストラクション（バイトコード）を実行するときに使用する変数を一時的に保存し、オペランドスタックとは、インストラクションのオペランド（引数）を積み上げたものをいう。具体的には、StackMap 属性には、指定されたバイトコードオフセットにあるローカル変数とオペランドスタックの型と個数を記録している。

（2）フェーズ 2：デバイス内の実行時検証

クラスファイルを携帯電話機器にダウンロードして、クラスファイルベリファイアにかける。クラスファイルベリファイアでは以下の動作を行う。

③ フォーマット検証

ロードされたクラスファイルについて、不正な項目が付加されていないか、必要な情報が欠けていないか等、クラスファイルが基本的なフォーマットを満たしているかどうかを調べる。

④ 項目の検証

クラスファイルで定義されている各項目のうち、次の③で検証する箇所以外のすべての項目について矛盾がないか、項目の一貫性を調べる。

⑤ バイトコードベリファイ

クラスファイルの中でメソッドを実行するためのインストラクションバイトコードを調べ、どのようなデータフローで実行されたとしてもオペランドスタックやローカル変数がオーバーフローやアンダーフローすることはないか、ローカル変数の値が設定される前にアクセスされることがないか、インストラクションの期待する型の値が格納されるかなど、インストラクションの正当性を検証する。

⑥ 実行時検証

他のクラスを参照しているバイトコードの場合に、そのクラスの情報との間に矛盾がないか、クラスファイルとの一貫性を調べる。クラスファイルのロード時に参照されるすべてのクラスについてベリファイするのは効率が悪いため、参照されるクラスについてはリンク時に検証を行う。

J2SE のベリファイアと CLDC のベリファイアの大きな違いは、バイトコードベリファイアの仕組み（⑤の実行時）である。通常のバイトコードベリファイアでは、対話的にデータフロー解析を行ってから検証を行うのに対し、CLDC のバイトコードベリファイアでは、事前検証で追加されている StackMap 属性を利用することで、バイトコードを先頭から最後まで 1 回の走査で検証することができる。そのため、通常のバイトコードベリファイアに比べ、データフロー解析にかかる時間とメモリを抑えることができる。

この点を除けば、本質的には標準の JavaVM と同等のベリファイを行うので、セキュリティ面においても異なる所はない。ベリファイアにパスしたクラスは妥当なものであり、メモリも破壊しないことになる。

4.1.5 アプリケーションの管理

クラスファイルベリファイアは、指定されたプログラムが妥当なクラスファイルであることを保証するが、ファイルシステムなどのリソースへのアクセスはベリファイアでは検証できないため、これだけではサンドボックスモデルを実現することはできない。

J2SE 環境では、セキュリティマネージャの概念を提供し、Java アプリケーションから外部リソースへのアクセスを規制している。CLDC では、メモリ消費を抑えるため、セキュリティマネージャを使用せず、単純な方法を取っている。

CLDC（KVM）では、Java Native Interface（JNI）を実装していない。JNI は、Java からネイティブな機能を使用するために、C 言語などの他の言語で書かれたアプリケーションや共有ライブラリを呼び出すためのインタフェースである。JNI をサポートしないため、ユーザが直接 JNI を使ってネイティブ機能にアクセスすることはできない。つまり、Java アプリケーションは、CLDC やプロファイルで事前に定義された API からのみ、ネイティブ機能を利用することができるようになっている。

JNI のサブセットとして K Native Interface（KNI）が定義されており、KVM に搭載することができるが、ユーザ定義のネイティブなプログラムを使用したり、共有ライブラリをロードすることはできない。

通常、アプリケーションからのリソースへのアクセスは、VM とは独立したコンポーネントの Java Application Manager（JAM）と呼ばれる機能を利用して行っている。

KVM では、移植の際にシステム構成オプションとして、JAM コンポーネントを含めるかどうかを指定することができる。KVM に付属の JAM は、アプリケーションのダウンロード、インストール、検査、起動、インストール解除を行うネイティブ C アプリケーションである。

Java アプリケーションの起動時は、JAM がアプリケーションのメインクラスを含むパラメータを使用して KVM を実行する。アプリケーションの実行のためにクラスがさらに必要な場合、KVM は事前に定義されている API を使用してクラスファイルのロードを行う。

以上に述べたように、CLDC のリファレンス実装である KVM は、通常の JavaVM のサブセットではあるが、通常の Java と同等の検証を少量のメモリで実現できるような仕組みや、セキュリティ上不完全な機能についてはサポートしないことにより、高度なセキュリティを

維持している。

4.1.6 情報資産の保護

情報処理振興事業協会（現：情報処理推進機構）平成 12 年度調査報告「Java を搭載する携帯端末機器におけるコンピュータウイルスの危険性に関する調査・検討結果報告書」において、KVM（Java 対応携帯電話）上でのウイルスの動作実験結果を報告している。ここでの報告にあるように、CLDC ライブラリではファイル操作機能がサポートされていないため、JVM で実行可能なファイル操作を行う不正な Java プログラムは KVM 上では動作させることができない。つまり、携帯電話上の送受信メールやアドレス帳データといったネイティブリソースへのアクセスは、KVM の機能だけでは行うことができない。ネイティブリソースへのアクセス機能が上位 API にて提供されている場合に限り、KVM はその API を利用したクラスファイルを実行し、アクセスすることができる。

VM に関しては、各機器メーカにおいて KVM が仕様どおりに実装されていれば、KVM の機能による情報の漏洩はないと考えてよい。

4.2　JBlend のセキュリティ調査

4.2.1　JBlend の特徴

JBlend は、アプリックス社が開発した組み込み向け Java プラットフォームの総称である。各種 Java の仕様へ対応しており、Sun Microsystems 社が提供するオリジナルのソースコードに基づいて作られている。主に携帯電話に搭載されているのは、CLDC に対応した JBlend[micro]（通称 microJBlend）である。

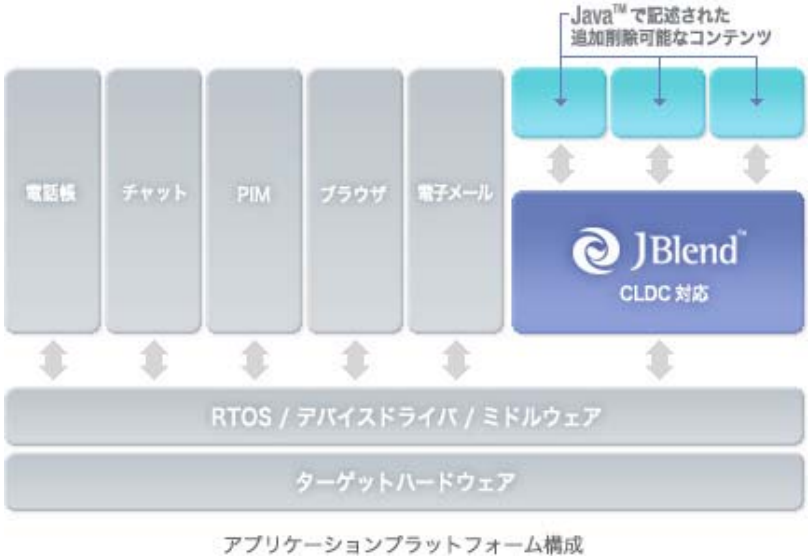

図 5　携帯電話における JBlend の位置付け
（http://www.jblend.com/JBlend/practical05.html より）

microJBlend は、以下のソフトウェアから構成される。

- JVM ライブラリ
  JVM の機能と CLDC クラスライブラリの機能をもつ。コンパイル済みの、リンク可能なライブラリとしてアプリックス社から提供する。
- 機器メーカ実装部
  JBlend を組み込むために機器メーカがプログラミングする部分。
- アプリケーション管理ソフトウェア
  MIDP 仕様に準拠した Java 実行環境には必須のプログラムで、製品の機能に合わせて機器メーカが作成する部分。以下のような機能を担当する。
  - 通信環境などからの Java アプリケーションのダウンロード
  - Java アプリケーションの機器へのインストール、アップデートおよび削除
  - Java アプリケーションの起動・終了および一時停止・再開

これらが作成されると、Java アプリケーションが実行できるようになる。JBlend では、これらをまとめて Java 実行環境と呼んでいる。

図 6　microJBlend のソフトウェア構成（http://www.jblend.com/lineup/micro02.htmlより）

JVM ライブラリには、以下のようなプロファイル、API、拡張プロファイルや、他の新規 JSR についても、簡単に追加することができる。

- MIDP 1.0/2.0
- DoJa
- J2ME Wireless Messaging API
- J2ME Mobile Media API
- JSCL
- KDDI-P
- VSCL

また、JVM ライブラリ自身も CLDC1.0/1.1 CLDC-HI 対応品に差し替えることができる。

4.2.2　Java 仕様との対応

JBlend は前述したように標準 Java のオリジナルコードを使っている。つまり、KVM 仕様の実装である。JavaVM の動作は基本的に CLDC の規定に従うが、以下の点において独自のサポートを行っている。

- KFTT による KVM 動作の高速化
  独自のソフトウェアによる高速化技術「KFTT」を利用することで、KVM の高速動作を実現している。
- ネイティブメソッドインタフェースの対応
  オプションの「外部クラスライブラリフレームワーク」を追加することで、Java アプリケーションからネイティブコードを呼び出すことができる。インタフェースには、KNI、アプリックス独自のネイティブメソッドインタフェース（KNI よりシンプルな設計）のどちらかを選択する。

さらに、セキュリティ面においては、以下の独自サポートを行っている。

- 内部パッケージ、クラス、メソッド名の隠蔽と使用の禁止
  非公開のパッケージ名、クラス名、メソッド名については、その名称を隠蔽する。また、アプリケーション起動時にクラスファイルの内容をチェックし、非公開のパッケージが含まれていた場合、アプリケーションを起動できない。
- コール元のパッケージ名確認
  セキュリティモデルでは、セキュリティポリシーを仲介させることで、その安全性を保とうとするが、セキュリティポリシーを仲介せずに実行できた場合、安全性を保つことができない。そこで、あるメソッドをコールしたメソッドのパッケージ名をチェックし、セキュリティポリシーのチェックをパスしていないかの検証を実行時に行う。
- クラスパス制限
  特定プロファイル用に設計されたパッケージは、他のプロファイルの仕様に基づいたアプリケーションからは見えない。

CLDC セキュリティモデルの実現のためには、以下のような方法がとられている。

- システムクラスのプリリンク
  CLDC の仕様では、システムクラスがオーバーライドできないよう規定している。そこで、JBlend では、システムクラスをプリリンク（ROM 化）することで、これを実現している。また、ユーザがシステムクラスと同じ名前のクラスを作成してアプリケーションに実装しても、システムクラスだけを使用し、なりすましや置き換えを防いでいる。

4.2.3 CLDC ライブラリの実装

CLDC ライブラリは、J2SE の API のうち、java.lang、java.util、java.io の 3 つと、CLDC 独自の javax.microedition.io の全部で 4 つのパッケージ構成となっている。実装においては、これらすべてにおいて、CLDC 1.0 仕様書に準拠したパッケージを JVM ライブラリに含んでいる。CLDC 1.1 仕様に準拠したパッケージも対応予定のようである。

4.2.4 MIDP ライブラリの実装

JVM ライブラリでは、MIDP 1.0/2.0 に準拠したパッケージを追加することができる。これを実装することにより、MIDP 1.0/2.0 仕様に基づいた Java アプリケーションを実行することができる。

4.2.5 DoJa ライブラリの実装

JVM ライブラリでは、DoJa 1.0/2.0/3.0 に準拠したパッケージを追加することができる。これを実装することにより、NTT ドコモの i モード対応 Java コンテンツを実行することができる。

4.2.6 拡張プロファイルの実装

JVM ライブラリでは、vodafone 独自の拡張 API 群である JSCL や、KDDI 独自の拡張 API 群である KDDI-P を追加することができる。これらを実装することにより、vodafone の V アプリや au の EZ アプリを実行することができる。また、各独自パッケージは、他のプロファイルからは使用できない仕組みが組み込まれており、他のアプリケーションを起動してもそれらのパッケージは見えない。

また、「外部クラスライブラリフレームワーク」を組み込むことで、JVM ライブラリに含まれない機器メーカ独自のクラスライブラリやサードパーティ製のクラスライブラリなどを追加することもできる。

vodafone では、アプリックス社と共同で開発した拡張 JBlend を全端末共通で採用している。拡張 JBlend についての詳細は公開されていないが、microJBlend をコアに、JVM ライブラリに MIDP と JSCL を実装したものである。vodafone では、JSCL は CLDC や MIDP の仕様を変更するものではないとしているため、microJBlend と拡張 JBlend との間で JavaVM の機能に違いはないと考えられる。

同様に、KDDI では、JVM ライブラリに MIDP と KDDI-P を実装した JBlend（KDDI 向け microJBlend）を採用している。JavaVM の機能は、vodafone と同様に標準の microJBlend と同等の機能を持つと考えられるが、API レベルにおいては、MIDP との違いを以下のように指摘している。

- NotifyPaused()の挙動【Phase1/2 対応端末】
  MIDP 仕様では NotifyPaused()を実行することにより中断状態になるが、KDDI-Java 環境では中断状態で止まらずに即座に再開してしまう。（Phase2.5 対応端末は MIDP 仕様どおりに動作する。）
- 標準時間の扱い
  J2ME Wireless Tool Kit でサポートしているタイムゾーン（GMT、UTC）に加え、JST も対応している。
- クリアキー押下
  MIDP 仕様ではクリアキーはないが、KDDI-Java 環境ではクリアキー押下（keyPressed()、keyReleased()、keyRepeated()）を取得することができる。ただし、クリアキー押下を getGameAction(int keyCode)にて処理させようとすると、getGameAction(int keyCode)が IlleagalArgumentException を送出する。

- 中断時の処理

  IMAP4MailConnection クラス、PhoneConnection クラスなどを利用しているときに一時中断を実行した場合には中断状態になり、再開を実行すると中断したところから処理を再開する。

4.2.7 JAM の実装

JBlend では、JAM は「アプリケーション管理ソフトウェア」として機器メーカの実装部分としており、アプリックスでは提供していない。各機器メーカが各国内通信キャリアの仕様に従い実装している。

4.2.8 エミュレータによる検証

JBlend の動作を確認するため、今回は Java 実行環境としてアプリックス社提供の CLDC 1.0 仕様 microJBlend デバイスエミュレータ、KDDI 提供の ezplus Emulator、vodafone 提供の J-SKY Application Emulator を用いて、Windows2000 上で実験を行った。

実験では、microJBlend が CLDC セキュリティモデルに従いサンドボックスモデルを実現しているかどうかを調査するため、以下の 3 点について検証した。

（1） クラスファイルベリファイアを利用して Java クラスファイルが不正でないことを検証しているか
（2） 事前に定義された API 以外を利用することができないか
（3） メモリを消費し続けるプログラムを実行するとどのような動作をするか

（1）クラスファイルベリファイアの実行による検証

通常、携帯電話用のアプリケーションは、以下の手順で開発する。

① テキストエディタによるソースプログラムの作成（.java）
② J2SE SDK を用いたコンパイルによるクラスファイルの生成（.class）
※ 各プロファイルを CLASSPATH に指定する
③ J2ME Wireless Toolkit を用いた事前検証済み（インライン展開および StackMap 属性付）クラスファイルの生成（.class）
④ J2SE SDK を用いてクラスファイルやリソースファイルをまとめた jar ファイルの作成（.jar）
⑤ jar ファイルの特性を記述したテキストファイルの作成
※ jar ファイルの特性を記述したファイルの拡張子や、その他に必要なファイルなどは、国内通信キャリアにより多少異なる。

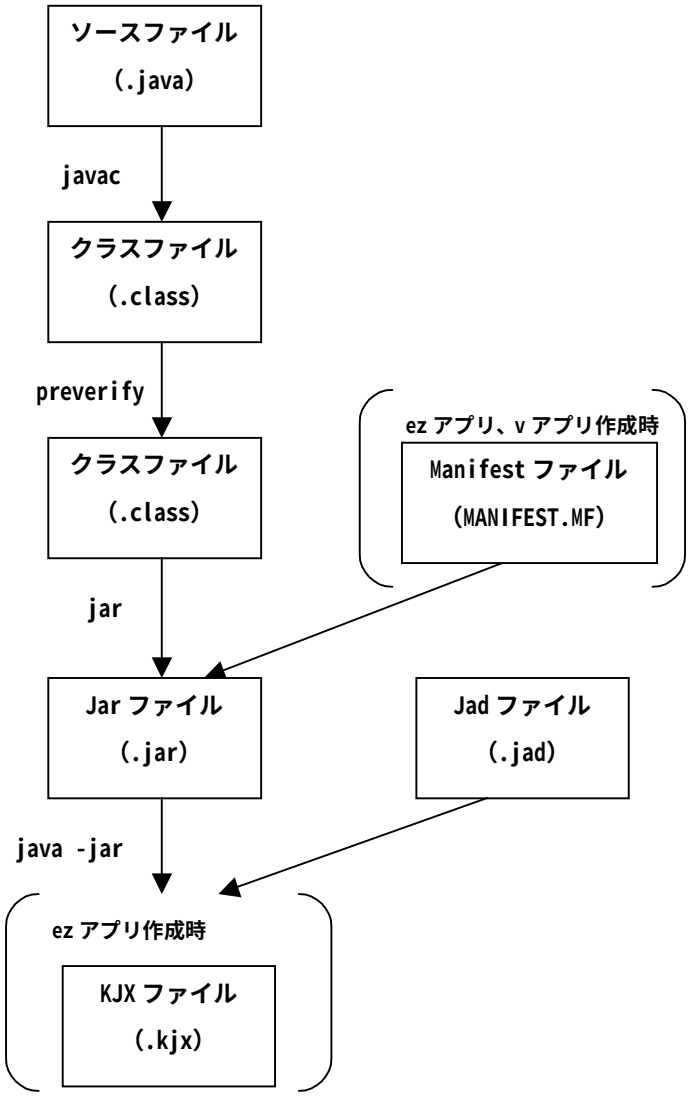

図 7　携帯電話用アプリケーションの作成手順例

まずは、各エミュレータに添付のサンプル Java ソースプログラムをビルドする際に、事前検証（手順③）を実行していないクラスファイルを jar ファイルにしてエミュレータ上で起動してみた。

- microJBlend デバイスエミュレータによる結果

microJBlend デバイスエミュレータに添付のサンプルソースプログラムは、一般的に Java プログラムの最初に作成するプログラムとしてよく知られる HelloWorld.java である。今回は、以下のようなソースプログラム（HelloWorldRP.java）を用いて実験を行った。

```
public class HelloWorldRP {
	public HelloWorldRP() {

		for (int i = 0;i<5; i++) {
			System.out.println("Hello World!");
		}

	}
	public static void main(String arg[]) {
		HelloWorldRP hw = new HelloWorldRP();
	}
}
```

図 8　HelloWorldRP ソースプログラム

通常の手順で、事前検証後のクラスファイルを圧縮した jar ファイルを実行すると、「HelloWorld!」の文字が 5 回実行画面上に表示される。

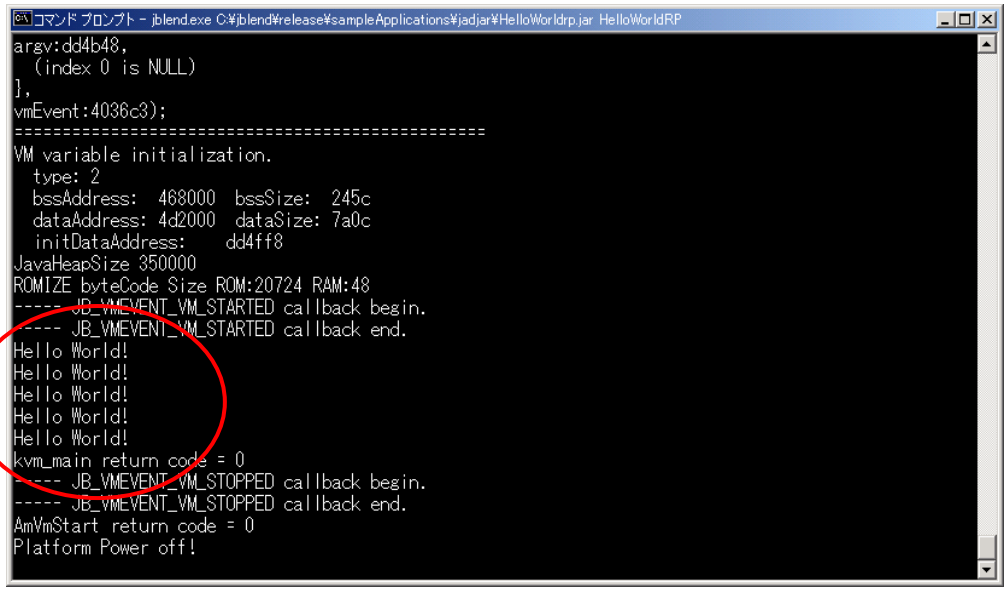

図 9　通常のビルド方法による HelloWorldRP 実行結果

HelloWorldRP.Java をコンパイルして作成された HelloWorldRP.class（事前検証前）をそのまま jar ファイルにしたものをエミュレータ上で起動したところ、ベリファイエラーのメッセージが表示された。結果は以下のとおりである。

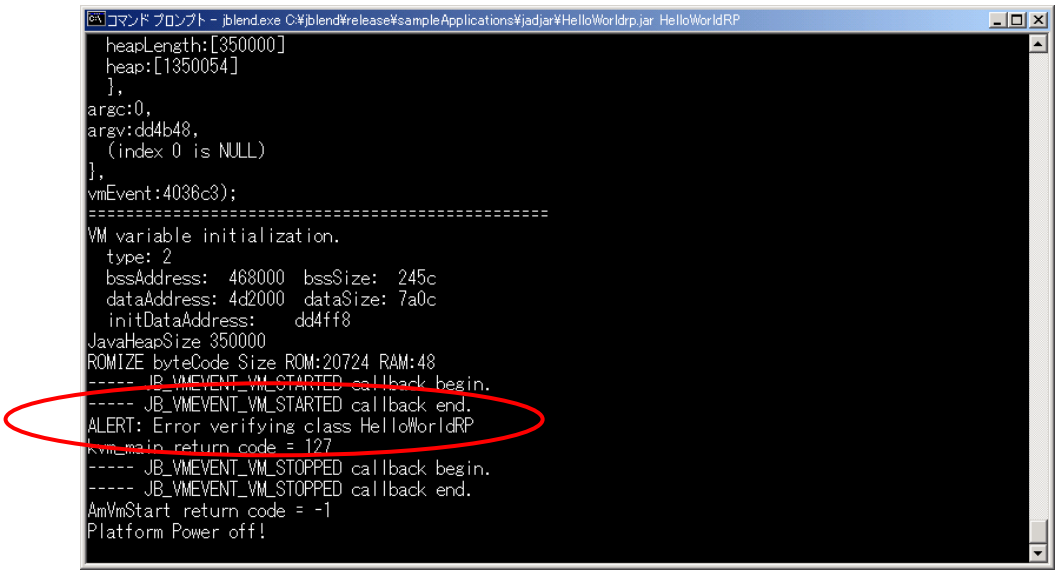

図 10　microJBlend デバイスエミュレータでの事前検証なしのクラスファイル実行結果

「ALERT: Error verifying class HelloWorldRP」というメッセージが表示され、アプリケーションが終了している。

このことから、microJBlend デバイスエミュレータの実行環境では、アプリケーションの起動時にはクラスファイルベリファイアにて事前検証済みのクラスファイルをチェックしていることが確認できた。

- ezplus Emulator による結果

ezplus Emulator にはサンプルとして 2 つのアプリケーションが同梱されている。このうち、実験には Adseesaw.java ソースプログラム（シーソーゲーム）を用いた。

事前検証していないクラスファイルをもとに作成した jar ファイルを使って、エミュレータ上で実行したところ、ベリファイエラーとなり、アプリケーションを実行することができなかった。結果は以下のとおりである。

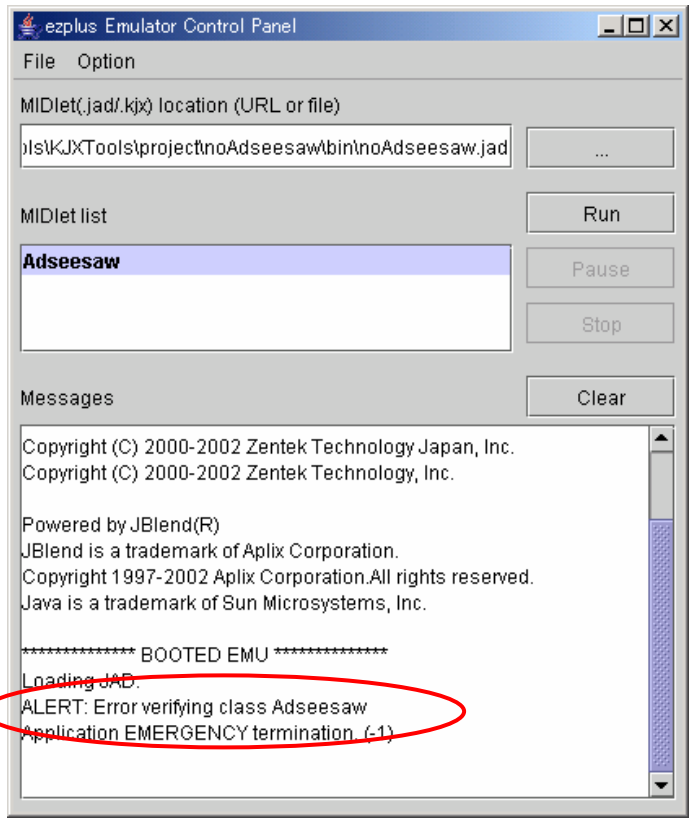

図 11　ezplus Emulator での事前検証なしのクラスファイル実行結果

「ALERT: Error verifying class Adseesaw」というメッセージが表示され、アプリケーションが終了している。

このことから、KDDI-Java 環境では、アプリケーションの起動時にはクラスファイルベリファイアにて事前検証済みのクラスファイルをチェックしていることが確認できた。

- J-SKY Application Emulator による結果

J-SKY Application Emulator には、サンプルアプリケーションは添付されていないため、Web 上に公開されていた「HelloWorld!!」と画面上に表示する MIDP アプリケーションのソースプログラム（HelloWorld.java）を利用した。

（http://www.zdnet.co.jp/mobile/0203/08/n_j1_2.html より）

事前検証していないクラスファイルをもとに作成した jar ファイルを使って、エミュレータ上で実行したところ、ezplus Emulator と同様にベリファイエラーとなり、アプリケーションを実行することができなかった。結果は以下のとおりである。

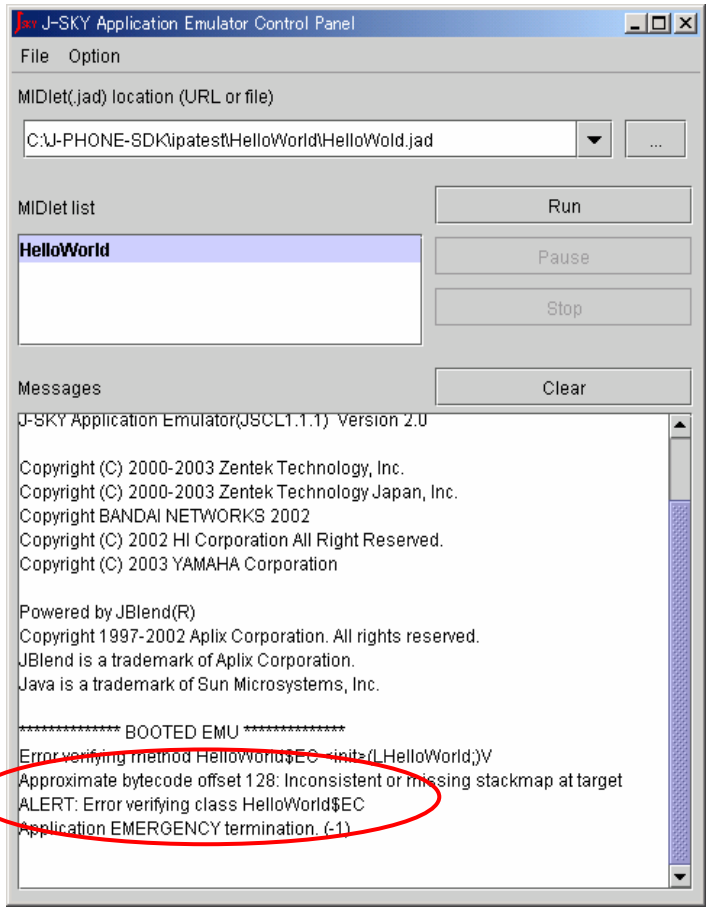
J-SKY Application Emulator Control Panel
File Option
MIDlet(.jad) location (URL or file)
C:\J-PHONE-SDK\ipatest\HelloWorld\HelloWold.jad
...
MIDlet list
Run
HelloWorld
Pause
Stop
Messages
Clear
J-SKY Application Emulator(JSCL1.1.1) Version 2.0
Copyright (C) 2000-2003 Zentek Technology, Inc.
Copyright (C) 2000-2003 Zentek Technology Japan, Inc.
Copyright BANDAI NETWORKS 2002
Copyright (C) 2002 HI Corporation All Right Reserved.
Copyright (C) 2003 YAMAHA Corporation
Powered by JBlend(R)
Copyright 1997-2002 Aplix Corporation. All rights reserved.
JBlend is a trademark of Aplix Corporation.
Java is a trademark of Sun Microsystems, Inc.
********** BOOTED EMU **********
Approximate bytecode offset 128: Inconsistent or missing stackmap at target
ALERT: Error verifying class HelloWorld$EC
Application EMERGENCY termination. (-1)

図 12　J-SKY Application Emulator での事前検証なしのクラスファイル実行結果

「ALERT: Error verifying class HelloWorld$EC」というメッセージが表示され、アプリケーションが終了している。このことから、V アプリの実行環境では、アプリケーションの起動時にはクラスファイルベリファイアにて事前検証済みのクラスファイルをチェックしていることが確認できた。

次に、事前検証後のクラスファイルをバイトコードレベルで改ざんし、改ざんしたクラスファイルを実行できるか試みた。実験に使ったソースプログラムは以下のような簡単な足し算を行うものである。

```
class Add {
	public static void main(String args[]) {
		try {
			keisan(10,10);
		} finally {
			System.out.println("end");
		}
	}

	static void keisan(int i,int j){
		int ans = i + j;
		System.out.println( i + " + " + j + " = " + ans );
	}
}
```

図 13　改ざん用ソースプログラム（Add.java）

実験では、次の 3 箇所について改ざんを行った。

① 引数として与える「10」を「1」に改ざん
② 足し算（iadd）コードを引き算（isub）コードに改ざん
③ 2 つめの「10」をスタックに積む（bipush）コードをすべて取り出す（pop）コードに改ざん（bipush コード 1 つに対して pop コード 2 つを実行）

- microJBlend デバイスエミュレータによる結果

①、②は改ざん後の計算結果が表示される結果となった。

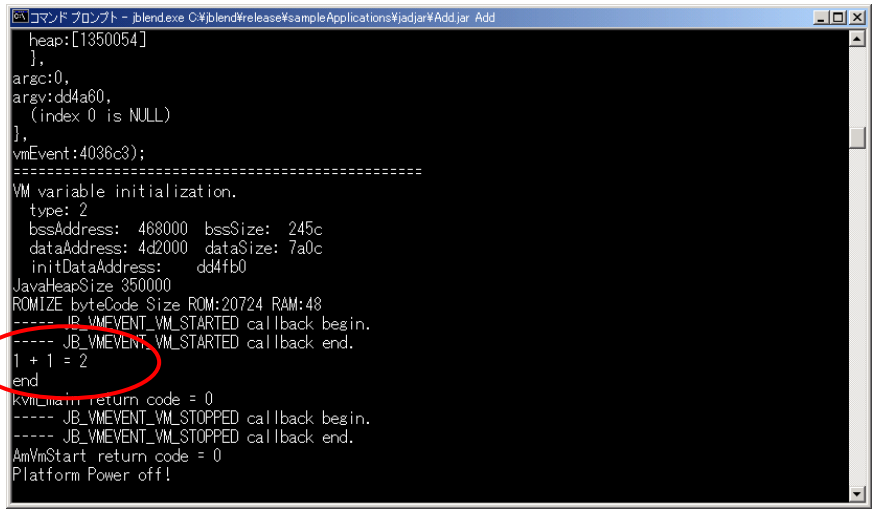

図 14　①実行後の Add の結果

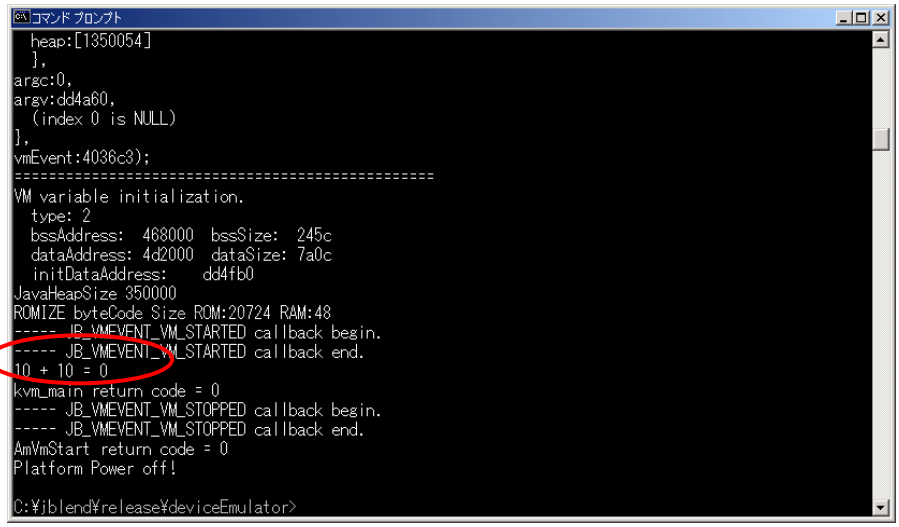

図 15　②実行後の Add の結果

③では、ベリファイエラーのメッセージが表示され、アプリケーションが終了した。

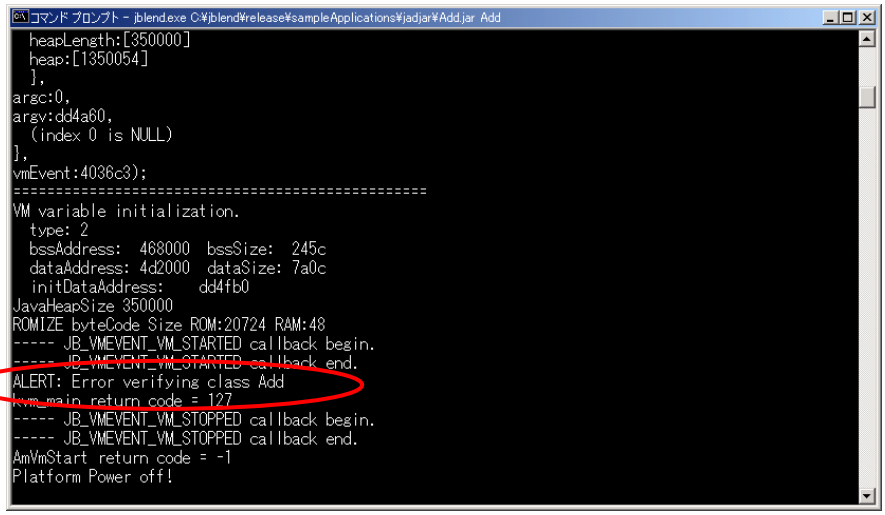

図 16　③実行後の Add の結果

J2SE 環境で実行すると、java.lang.VerifyError クラスが呼び出される。しかし、J2SE で定義されているエラークラスのほとんどは CLDC の仕様から削除されており、J2SE 環境での実行時とは異なる結果となった。ここでは「ALERT: Error verifying class Add」が表示され、アプリケーションが終了している。

以上のことから、クラスファイルをバイトコードレベルで改ざんした場合でも、サンドボックスモデルは有効に働き、Java 実行環境が暴走したり他のメモリ領域を参照するといったことが無いことは確認できた。しかし、数値や演算命令などのバイトコードでの改ざんに対しては、Java 実行環境はその防護策を持たず、より上位のセキュリティモデル、例えばデジタル署名を利用した保護などが必要なことも同時に確認できた。

- ezplus Emulator による結果

ezplus Emulator では、J2SE 用の main メソッドで始まるソースプログラムでは実行できないため、Add.java を MIDlet アプリケーションのソースプログラムに変更した。このソースプログラムをコンパイルし、preverify を行ったクラスファイルを生成し、①、②、③について実験した。

①、②は、改ざん後の計算結果が表示される結果となった。

図 17　②実行後の Add の結果

③では、ベリファイエラーのメッセージが表示され、アプリケーションが終了した。

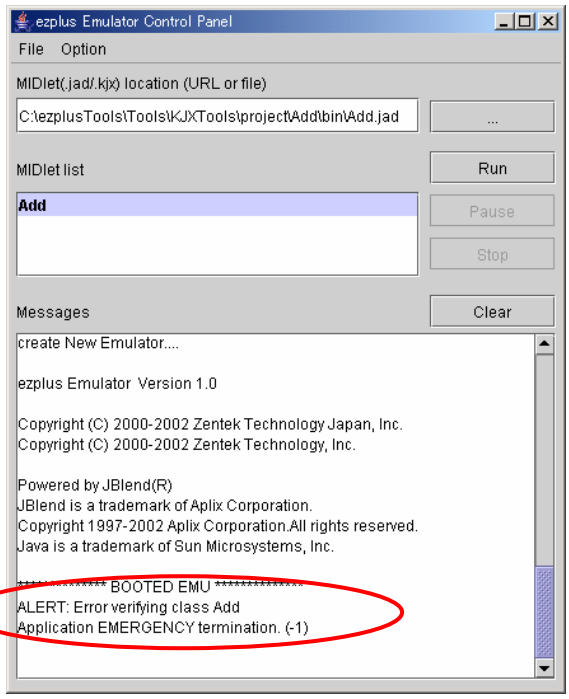

図 18　③実行後の Add の結果

microJBlend デバイスエミュレータの結果と同様、「ALERT: Error verifying class Add」というメッセージが表示され、アプリケーションが終了している。

以上のことから、EZ アプリについても、クラスファイルをバイトコードレベルで改ざんした場合でも、サンドボックスモデルは有効に働き、Java 実行環境が暴走したり他のメモリ領域を参照するといったことが無いことは確認できた。しかし、数値や演算命令などのバイトコードでの改ざんに対しては、Java 実行環境はその防護策を持たず、より上位のセキュリティモデル（例えばデジタル署名を利用した保護など）が必要なことも同時に確認できた。

- J-SKY Application Emulator による結果

J-SKY Application Emulator も、J2SE 用の main メソッドで始まるソースプログラムでは実行できないため、ezplus Emulator で使用したものと全く同じ MIDlet アプリケーションのクラスファイルを使用して、①、②、③について実験した。

①、②は、改ざん後の計算結果が表示される結果となった。

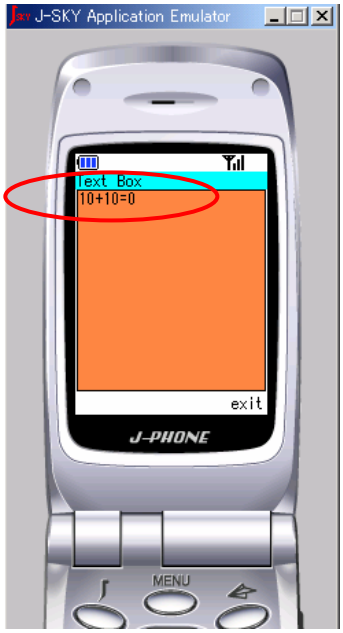

図 19　②実行後の Add の結果

③では、ベリファイエラーのメッセージが表示され、アプリケーションが終了した。

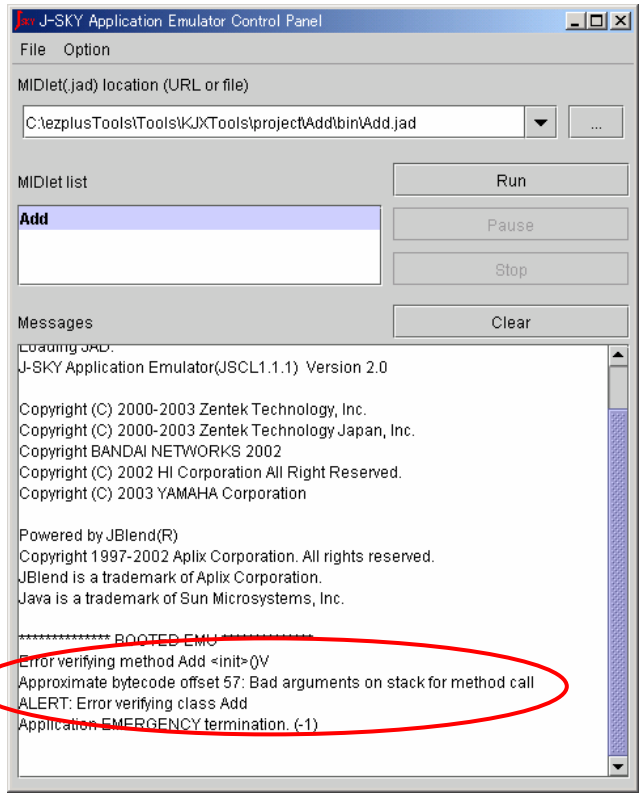

図 20　③実行後の Add の結果

他の 2 つのエミュレータの結果と同様、「ALERT: Error verifying class Add」というメッセージが表示され、アプリケーションが終了している。また、ここではさらに詳細に「Bad arguments on stack for method call」というメッセージも表示し、オペランドスタックの操作コードがおかしいことをベリファイ時に検出していることがわかる。

以上のことから、V アプリについても、クラスファイルをバイトコードレベルで改ざんした場合でも、サンドボックスモデルは有効に働き、Java 実行環境が暴走したり他のメモリ領域を参照するといったことが無いことは確認できた。しかし、数値や演算命令などのバイトコードでの改ざんに対しては、Java 実行環境はその防護策を持たず、より上位のセキュリティモデル（例えばデジタル署名を利用した保護など）が必要なことも同時に確認できた。

（2）事前定義された API 以外の API の利用

CLDC ライブラリでは、J2SE で提供されている API の一部しか提供されていない。ここでは、CLDC では提供されていないクラスを利用したサンプルソースプログラムを作成し、各国内通信キャリアのビルド手順に従って、アプリケーションの作成、実行を試みた。

- microJBlend デバイスエミュレータによる結果

CLDC では提供されていないユーティリティのクラスを利用した J2SE 用のサンプルソースプログラムを使用した。java.util.Locale クラス を使用して、Java を実行したシェルのロケールを取得するコードである。

（Sun Microsystems　Java センター　サンプルコード集より引用
http://javacenter.sun.co.jp/java-sample/java.util/java_util_01.html）

```
import java.util.Locale;

public class LocaleTest {
    public static void main(String args[]) {
        // デフォルトロケールの取得
        Locale locale = Locale.getDefault();

        System.out.println("DisplayCountry : " + locale.getDisplayCountry());
        System.out.println("DisplayLanguage : " +
locale.getDisplayLanguage());
        System.out.println("Language : " + locale.getLanguage());
        System.out.println("DisplayName : " + locale.getDisplayName());
        System.out.println("Locale : " + locale);
    }
}
```

図 21　ロケール取得プログラム（LocaleTest.java）

J2SE 環境でコンパイル、実行すると、ロケール情報を画面上に表示する。

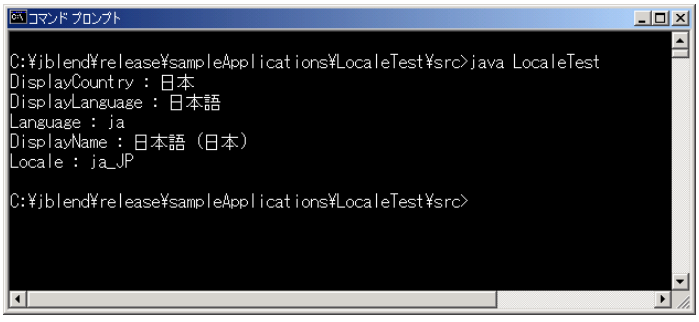

図 22　J2SE 環境での LocaleTest.class 実行結果

microJBlend デバイスエミュレータのビルド方法に従ってアプリケーションを作成すると、コンパイルの時点でエラーが発生し、class ファイルを作成することができなかった。

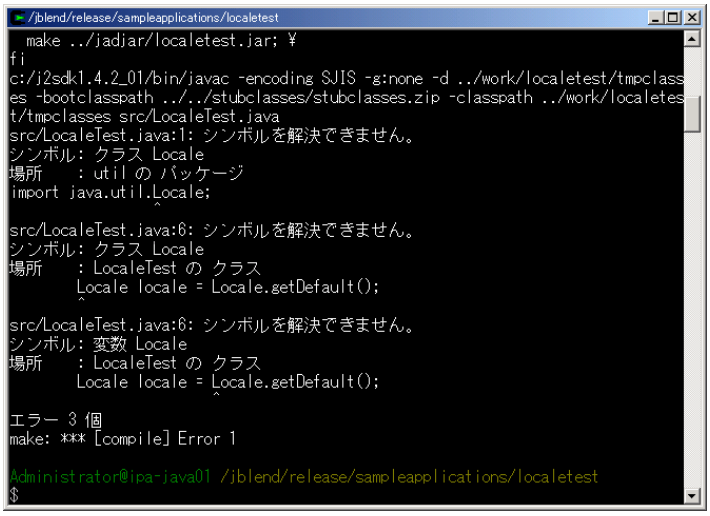

図 23　microJBlend デバイスエミュレータ用のサンプルアプリケーションビルド結果

Locale クラスを記述した箇所について、「シンボルを解決できません。」というコンパイルエラーが発生した。

また、J2SE 環境にてコンパイルに成功した LocaleTest.class を用いて preverify を実行し、生成されたクラスファイルから jar ファイルを作成した。この jar ファイルを使ってエミュレータ上でアプリケーションを実行したところ、やはりエラーが発生し、実行することができなかった。

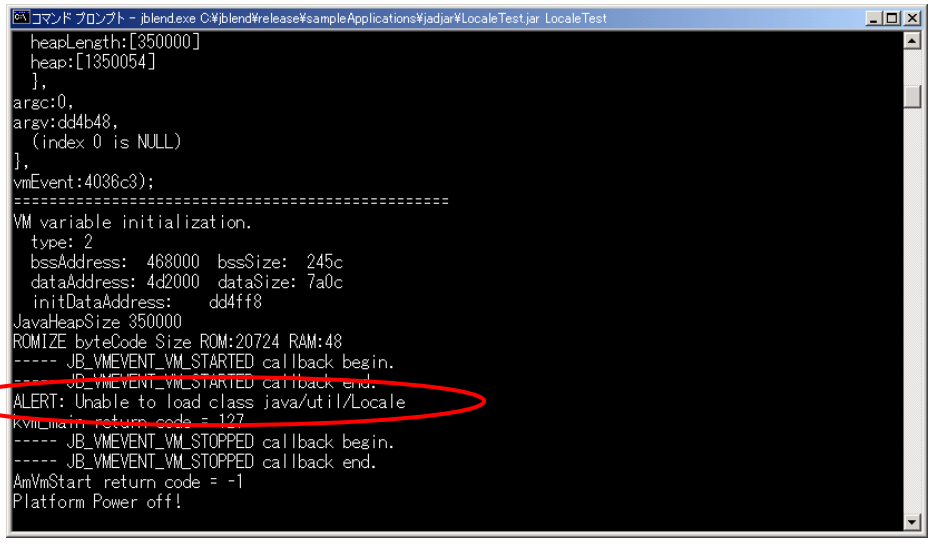

図 24　microJBlend デバイスエミュレータでの LocaleTest 実行結果

「ALERT: Unable to load class java/util/Locale」というメッセージが表示され、終了している。サポートしていないライブラリを呼び出したため、ロードできずにエラーとなる。

以上のことから、microJBlend デバイスエミュレータの実行環境では、サポートしていない API を使った java アプリケーションの開発や実行はできないことが確認できた。

- ezplus Emulator、J-SKY Application Emulator による結果

ezplus Emulator と J-SKY Application Emulator では、J2SE 用の main メソッドで始まるソースプログラムでは実行できないため、LocaleTest.java を MIDlet アプリケーションのソースプログラムに変更して実験を行った。

MIDlet アプリケーションのビルド方法に従ってアプリケーションを作成すると、コンパイルの時点でエラーが発生し、class ファイルを作成することができなかった。

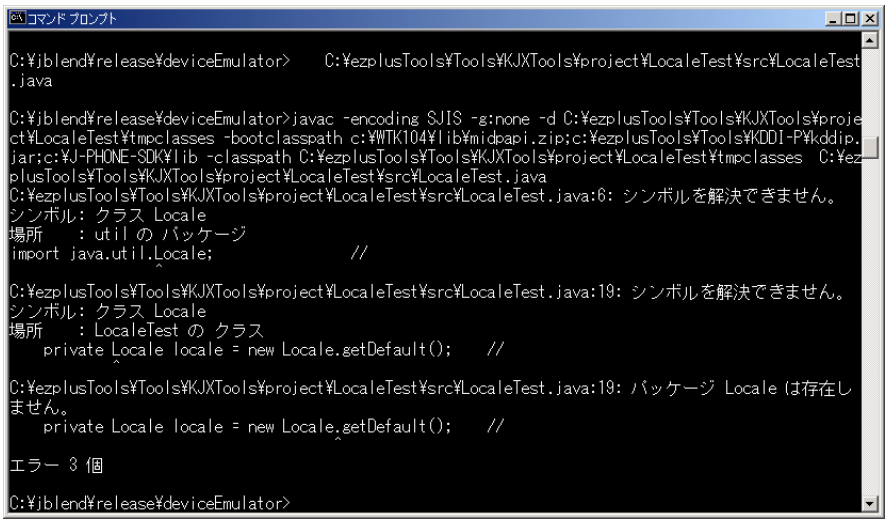
コマンド プロンプト
C:¥jblend¥release¥deviceEmulator>　C:¥ezplusTools¥Tools¥KJXTools¥project¥LocaleTest¥src¥LocaleTest.java
C:¥jblend¥release¥deviceEmulator>javac -encoding SJIS -g:none -d C:¥ezplusTools¥Tools¥KJXTools¥project¥LocaleTest¥tmpclasses -bootclasspath c:¥WTK104¥lib¥midpapi.zip;c:¥ezplusTools¥Tools¥KDDI-P¥kddip.jar;c:¥J-PHONE-SDK¥lib -classpath C:¥ezplusTools¥Tools¥KJXTools¥project¥LocaleTest¥tmpclasses C:¥ezplusTools¥Tools¥KJXTools¥project¥LocaleTest¥src¥LocaleTest.java
C:¥ezplusTools¥Tools¥KJXTools¥project¥LocaleTest¥src¥LocaleTest.java:6: シンボルを解決できません。
シンボル: クラス Locale
場所　: util の パッケージ
import java.util.Locale;　　//
C:¥ezplusTools¥Tools¥KJXTools¥project¥LocaleTest¥src¥LocaleTest.java:19: シンボルを解決できません。
シンボル: クラス Locale
場所　: LocaleTest の クラス
private Locale locale = new Locale.getDefault();　//
C:¥ezplusTools¥Tools¥KJXTools¥project¥LocaleTest¥src¥LocaleTest.java:19: パッケージ Locale は存在しません。
private Locale locale = new Locale.getDefault();　//
エラー 3 個
C:¥jblend¥release¥deviceEmulator>

図 25　ezplus Emulator 用のサンプルアプリケーションビルド結果

Locale クラスを記述した箇所について、「シンボルを解決できません。」というコンパイルエラーが発生した。

以上のことから、MIDlet アプリケーションの開発、実行環境では、サポートしていない API を使った java アプリケーションの開発や実行はできないことが確認できた。J2ME に存在しないはずの危険な行為が可能なクラスやメソッドは、やはり利用することができない。

（3）不正プログラム実行時の動作検証

クラスファイルベリファイアでは、オペランドスタックやローカル変数がオーバーフローやアンダーフローすることはないか等、インストラクションの正当性を検証している。

そこで、Java スタックのオーバーフローを起こす Java ソースプログラムや、ヒープ領域を消費し続ける Java ソースプログラムからアプリケーションを作成し、各エミュレータ上で動作させてみた。

```
class Fact {
        static int j =0;

        public static void main(String args[]) {
                Fact fc = new Fact();
                System.out.println("fact5="+ fc.fact(-1) );
        }
        public int fact(int i){
                if (i==0){
                        return 1;
                } else {
                        j++;
                        System.out.println("j="+ j );
                        return i * fact(i-1);
                }
        }
}
```

図 26　スタックオーバーフローさせるサンプルソースプログラム（Fact.java）
-1 の階乗計算

```
class Heep {
        static int j =0;
        static String res = "";

        public static void main(String args[]) {
                Heep hp = new Heep();
                hp.alloc();
        }
        public void alloc(){
                while (true){
                        String str = new String(" ");
                        res = res + str;
                        j++;
                        System.out.println("j="+ j );
                }
        }
}
```

図 27　ヒープ領域を使いつづけるサンプルソースプログラム（Heep.java）
文字列に 1 文字の追加を繰り返す

- microJBlend デバイスエミュレータによる結果

microJBlend デバイスエミュレータのビルド方法に従って、Fact.java と Heep.java をもとに Fact.jar、Heep.jar ファイルを作成した。これらの jar ファイルを使ってエミュレータ上でアプリケーションを実行したところ、どちらもメモリ不足エラーが発生してアプリケーションを終了した。

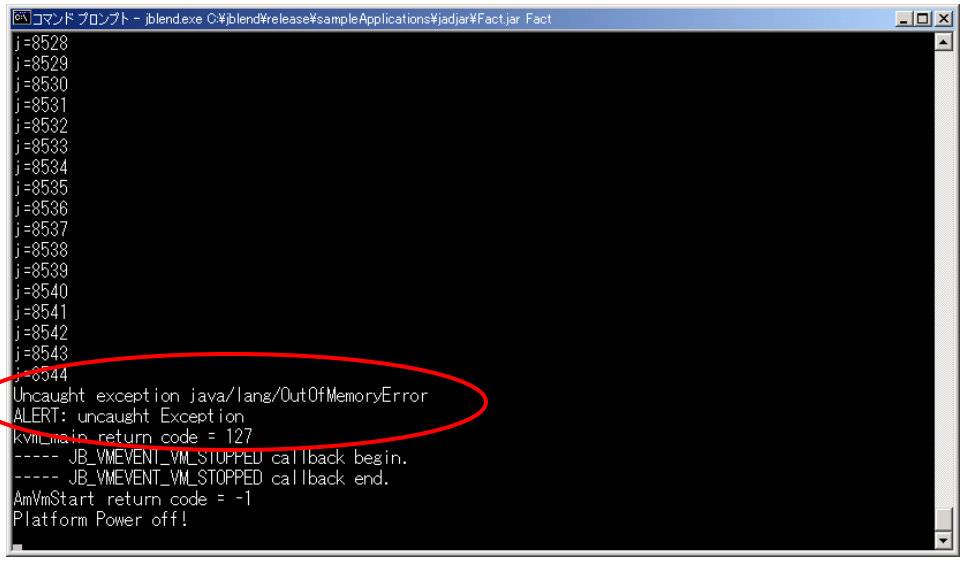

図 28　microJBlend エミュレータでの Fact 実行結果

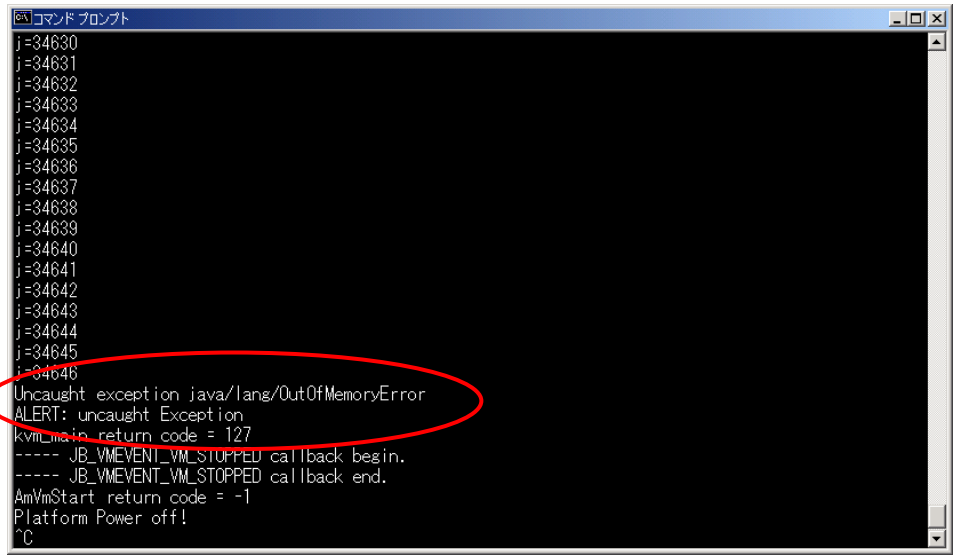

図 29　　microJBlend エミュレータでの Heep 実行結果

Fact.java を J2SE 環境でコンパイル、実行すると、java.lang.StackOverflowError クラスが呼び出されたが、今回は J2SE 環境での実行時のメッセージと異なり、どちらの場合も「Uncaught exception java/lang/OutOfMemoryError」となった。

J2SE で定義されているエラークラスのほとんどは、CLDC では仕様から削除されているため、ここでは java.lang.OutOfMemoryError クラスが呼び出されている。そのため、メッセージの内容からはスタックオーバーフローによるものかヒープ領域の不足によるものか、VM 中のどのメモリ領域についての例外かを特定することはできないが、VM に割り当てられている領域についてメモリ不足例外が確実に発生し、他のメモリ領域に侵害しないよう動作することが確認できた。

- ezplus Emulator と J-SKY Application Emulator による結果

ezplus Emulator と J-SKY Application Emulator では、J2SE 用の main メソッドで始まるソースプログラムでは実行できないため、Fact.java と Heep.java を MIDlet アプリケーションのソースプログラムに変更して、両エミュレータ上で動作可能な MIDlet アプリを作成した（Fact、Heep)。これらのアプリケーションを両エミュレータ上で実行したところ、以下のような結果となった。

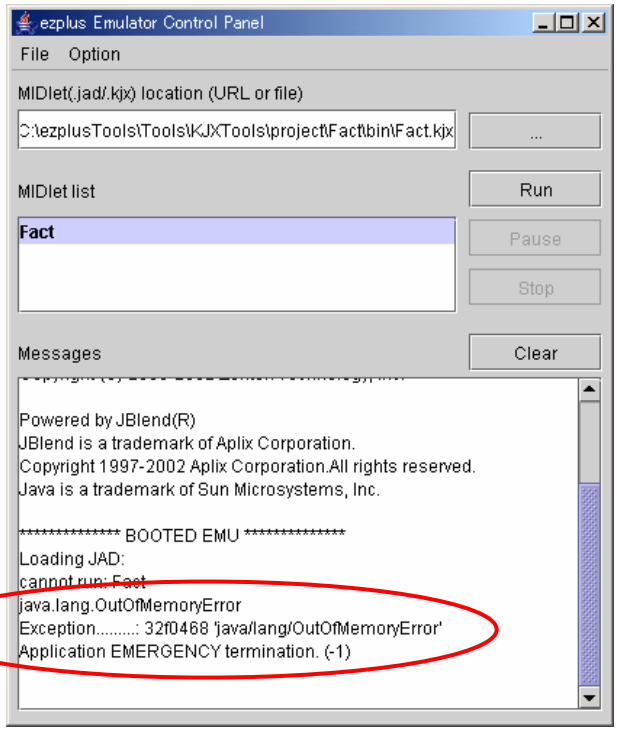

図 30　ezplus Emulator での Fact 実行結果

Fact アプリケーション実行時は、ezplus Emulator、J-SKY Application Emulator とも動作不安定となり、エミュレータのアプリケーションエラーが発生しエミュレータ自身が終了してしまうこともあり、常に同じ結果を得ることができなかった。エミュレータが異常終了しなかった場合は、ezplus Emulator では microJBlend デバイスエミュレータと同様に java.lang.OutOfMemoryError クラスが呼び出されて Fact が終了しているが、J-SKY Application Emulator では、異常終了を示す「Application EMERGENCY termination.」メッセージを表示することもなく、開始画面に戻ってしまった。

これらのことから、エミュレータ上では、スタックオーバーフロー時の動作には安定性がないことが確認された。

一方、Heep アプリケーションは、どちらのエミュレータ上でも常に java.lang.OutOfMemoryError クラスが呼び出され、アプリケーションを終了した。

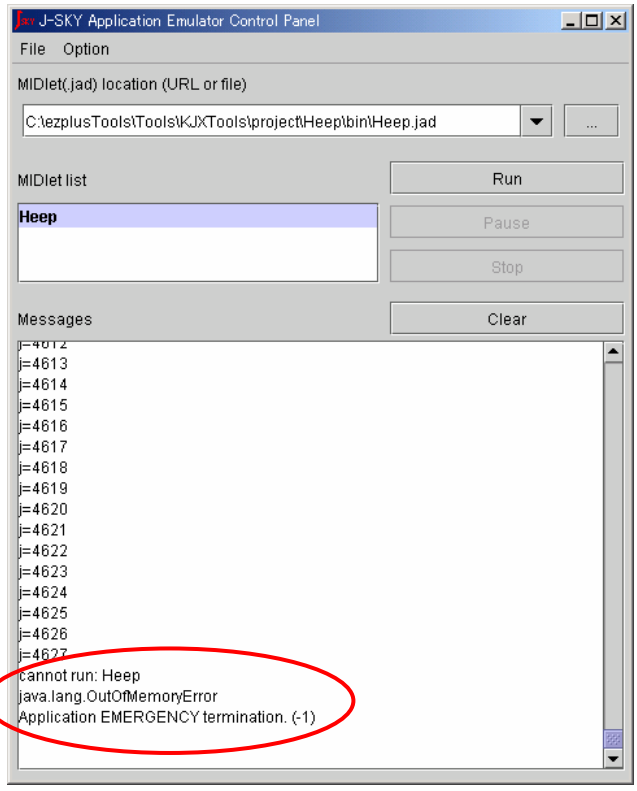

図 31　J-SKY Application Emulator での Heep 実行結果

この結果から、メモリを食いつぶしてもエラーとして安全に処理されることが確認できた。したがって、大量のメモリを消費しても他の機能に影響を与えることはないと考えられる。

4.2.9 情報資産の保護

microJBlend については、エミュレータによる検証の結果、クラスファイルベリファイアが機能していること、サポートしない不正な API は利用できないことが確認され、サンドボックスモデルが実現されていることが確認できた。

バイトコードレベルでデータ等を改ざんした場合はベリファイアでは検出することができないため、VM のレベルでは改ざんによる不正なアプリケーションが実行される可能性は残るが、それによってネイティブな機能に影響を与えることは難しく、情報資産が破壊、漏洩することはないと考えてよい。

4.3　JV-Lite2 のセキュリティ調査

4.3.1　JV-Lite2 の特徴

JV-Lite2 は、Sun Microsystems 社が認定する組み込み機器向け JavaVM である。独自実装の Java 互換 VM であった「JV-Lite」の実績をベースに開発された Java 実行環境であり、、Sun Microsystems 社の TCK（互換性テスト）により 100%の互換性を保証している。携帯電話などのリソースが限られた機器に搭載することを考慮して設計された製品として、ACCESS 社独自の JavaVM と ACCESS で独自に実装した Java クラスライブラリから構成される Wireless Edition を提供している。

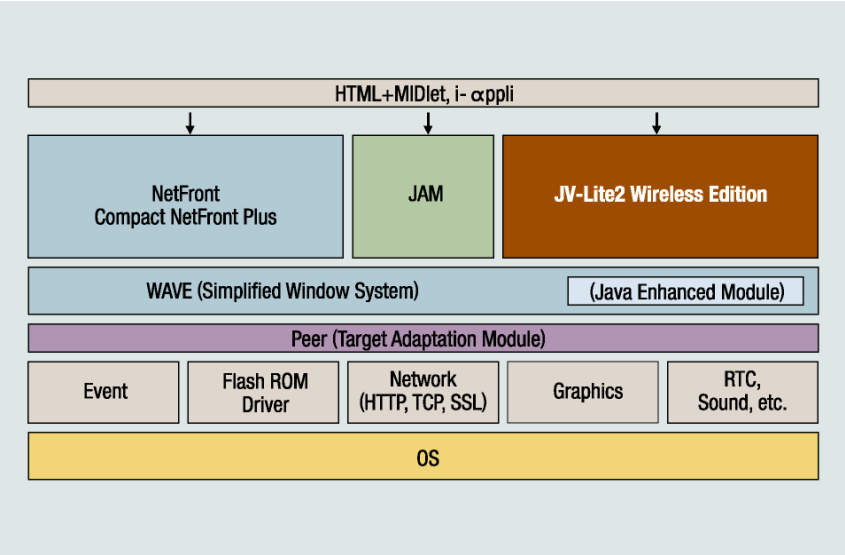

図 32　JV-Lite Wireless Edition（http://www.access.co.jp より）

JV-Lite2 Wireless Edition は以下のモジュールにより構成されている。

- JAM
  Java Application Manager。Java アプリケーションのダウンロードやダウンロードしたアプリケーションを管理するモジュール。
- JavaVM&CLDC
  CLDC 1.0 に準拠した JavaVM と CLDC ライブラリ。
- MIDP
  Moble Information Device Profile。Sun Microsystems 社が提案している携帯電話、ワイヤレス PDA 向けのプロファイル。
- DoJa（オプション）
  NTT ドコモが提案している CLDC をベースとした i モード対応 Java プロファイル。

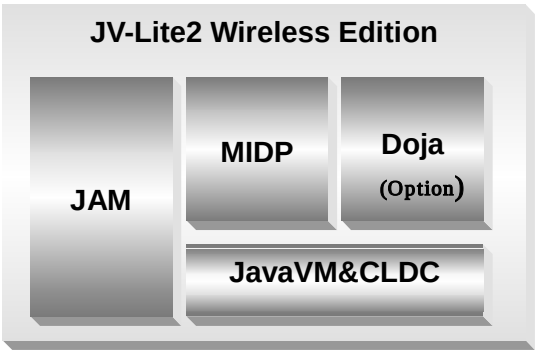

図 33　JV-Lite2 Wireless Edition の構成

JV-Lite2 Wireless Edition は、ACCESS 社の簡易ウィンドウシステム WAVE（Window-based Abstract Virtual Environment）上に実装される。WAVE は、ウィンドウシステムとしての機能、メモリ管理、イベントハンドリングなどの低レベルプログラムを提供し、ハードウェアや OS と上位モジュールの橋渡しを行う。これにより WAVE から上位のモジュールをハードウェアや OS から独立させ、JV-Lite2 Wireless Edition についても高い移植性を実現している。

JV-Lite2 Wireless Edition を WAVE 上に実装するためには、特定の Java 機能やファイルシステム機能などについて拡張が必要であった。そこで、それらの不足部分を補うモジュールとして WEJ（WAVE Extension for Java）が動作している。JV-Lite2 を携帯電話に適用する場合の構成例は以下のとおりである。

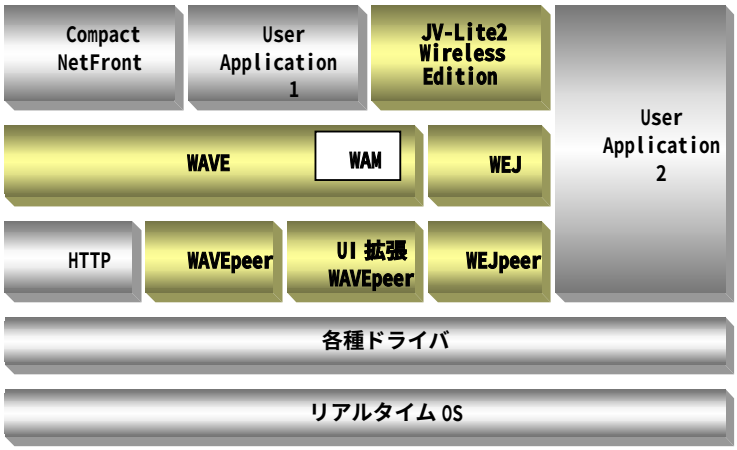

図 34　JV-Lite2 Wireless Edition の適用例

WAM（WAVE Application Manager）とは、WAVE 上で複数のアプリケーションを動作させるためのフレームワークである。

WAVE peer・UI 拡張 WAVE peer・WEJ peer は、WAVE、WEJ と低レベルドライバとのターゲットの依存部分を吸収する移植層になる。JV-Lite2 Wireless Edition を移植する場合は、この部分を開発して実装することが主な作業となる。JavaVM については、コードを変

更せずに設定を行うことができる。

4.3.2 Java 仕様との対応

JV-Lite2 Wireless Edition は、J2ME に準拠した Java 実行環境であり、基本ライブラリは CLDC 1.0 に準拠している。プロファイルとしては、MIDP 1.0a をサポートし、DoJa もオプションでサポートする。

JavaVM は CLDC 1.0 仕様に準拠し、以下の機能を削除している。

- 浮動小数点のサポート
- JNI のサポート
- ユーザ定義のクラスローダのサポート
- リフレクションのサポート
- スレッドグループとデーモンスレッドのサポート
- ファイナライズのサポート

4.3.3 CLDC ライブラリの実装

CLDC ライブラリは、J2SE の API のうち、java.lang、java.util、java.io の 3 つと、CLDC 独自の javax.microedition.io の全部で 4 つのパッケージ構成となっている。実装においては、これらすべてにおいて、CLDC 1.0 仕様書に準拠したパッケージを実装している。

ただし、仕様との相違点として、以下の 2 点が挙げられる。

1 点目は、java.io パッケージの実装である。CLDC 仕様では、エンコーディングのサポートは特に規定されていないが、実装ではエンコーディングとして ISO8859_1、UTF8、SJIS をサポートしている。これ以外のエンコーディングが指定された場合は、UnsupportedEncodingException をスローする。デフォルトエンコーディングは microedition.encoding プロパティに設定されており、設定はプロファイルに依存する。MIDP では ISO8859_1、DoJa では SJIS をデフォルトにしている。

2 点目は、javax.microedition.io パッケージの実装である。CLDC 仕様ではネットワーク接続に使用できるプロトコルは規定されておらず、どのようなプロトコルを指定できるかはプロファイルの実装に依存するが、実装では scheme として HTTP のみをサポートしている。HTTP プロトコルに特化した機能を定義するサブインタフェースがプロファイルで定義される。

4.3.4 MIDP ライブラリの実装

MIDP は、タイマー、ストレージ、HTTP 接続、ユーザインタフェースなどの機能があり、これらの機能はほとんど J2ME 用に用意された、javax.microedition.rms、javax.microedition.midlet、javax.microedition.io、javax.microedition.lcdui の４つのパッケージに含まれる。このほか、実行例外クラスを定義しているパッケージとして java.lang、タイマーに関するクラスを定義しているパッケージとして java.util を含め、６つすべてにお

いて MIDP 1.0a 仕様書に準拠したパッケージを実装している。

4.3.5 DoJa ライブラリの実装

DoJa は、MIDP とほぼ同等の機能をもつ NTT ドコモの独自仕様プロファイルである。JV-Lite2 Wireless Edition では、これをオプションとしてサポートしている。実装できるのは、DoJa 1.5 プロファイルで定義されている、com.nttdocomo.ui、com.nttdocomo.util、com.nttdocomo.lang、com.nttdocomo.io、com.nttdocomo.net の 5 つのパッケージである。すべて、「i モード対応 Java コンテンツガイド~API リファレンス編~1.1 版」に準拠したパッケージを実装している。

ただし、com.nttdocomo.ui パッケージについては、仕様との相違点もいくつかある。

１つめは、Graphics クラスである。このクラスは、最も基本的な描画機能を提供するクラスである。仕様では、FillPolygon メソッドは、頂点の数が 16 点以下の多角形について setColor メソッドで指定された色で塗りつぶし、それより多い頂点の多角形の場合の振る舞いは機種に依存するが、実装では頂点の数に制限を設けていない。また、Graphics オブジェクトがロックされた状態で参照がなくなった場合の状態は、仕様上は機種依存としているが、実装ではアンロックされたことと同じ状態になる。

２つめは、Image クラスである。このクラスは、イメージ（静止画像）を定義する。dispose はイメージを破棄するメソッドであり、仕様では dispose メソッドの呼出し後にそのオブジェクトに対してアクセスできないが、実装では MediaImage インタフェースの getImage メソッドによって戻された Image オブジェクトに対しては影響を与えない。

３つめは、Label クラスである。このクラスは、文字列を表示するためのコンポーネントを定義する。仕様では、コンポーネントサイズに文字列が収まらない場合の振る舞いは機種依存としているが、実装では右寄せで表示できる範囲を表示する。

４つめは、ImageLabel クラスである。このクラスはイメージを表示するためのコンポーネントを定義する。仕様ではイメージのサイズがコンポーネントのサイズより小さい場合の配置や大きい場合の振る舞いは機種依存としているが、実装では、小さい場合は中央に配置され、大きい場合は左上に揃える。

いずれにしても、主な相違点は、仕様上機種依存している部分について、実装では動作を明確にしているという点である。

4.3.6 JAM の実装

JV-Lite2 Wireless Edition の JAM は、MIDP 1.0a と「i モード対応 Java コンテンツ開発ガイド~API リファレンス編~1.1 版」に準拠している。

JV-Lite2 Wireless Edition の JAM は以下のような機能を持っている。

- プロファイルの識別

  複数のプロファイルに対応するため、ダウンロードされた ADF（Application Descriptor File）からダウンロードされるアプリケーションのプロファイルを識別する。ダウンロー

ドされた ADF ファイルの URL のコンテントタイプとその拡張子から識別する。

- ADF 解析

ブラウザによりダウンロードされた ADF の解析を行う。対応している ADF のキーは表51のとおりである。各キーの詳細は、MIDlet-Screen-Size、ScreenSize を除き MIDP 1.0a および「i モード対応 Java コンテンツ開発ガイド~API リファレンス編~1.1 版」に準拠している。プロファイルが MIDP の場合は、すべてのキーについて javax.microedition.midlet.MIDlet クラスの getAppProperty(String)メソッドを用いて情報を得ることができる。

MIDlet-Screen-Size、ScreenSize は、独自でアプリケーションの画面サイズを指定することができるように拡張している。指定サイズが実際の画面サイズ以下の時のみ効果を発揮する。

表 51　JV-Lite2 Wireless Edition 対応 ADF キー

Profile	File	Key	Type	備考
MIDP	ADF	MIDlet-Version	O	ダウンロード時の更新時にこの情報を元に判定する。この情報がない場合は、無条件に更新する。
		MIDlet-Jar-URL	M	この情報をもとにアプリケーションのダウンロードを行う。
		MIDlet-Name	O	この情報をもとにアプリケーション一覧で名前が表示される。この情報が設定されていない場合は、"???"と表示される。
		MIDlet-Jar-Size	M	この情報をもとにアプリケーションの保存領域を確保する。
		MIDlet-Data-Size	O	この情報をもとにデータ記憶域の最低必要サイズを確保する
	ADF または Manifest File	MIDlet-<n>	M	この情報をもとに JavaVM 起動後のクラス起動の動作が決定する。
		MIDlet-Screen-Size	O	この情報をもとに MIDP の画面サイズを決定する。
DoJa	ADF	AppName	O	この情報をもとにアプリケーション一覧で名前が表示される。この情報が設定されていない場合は、"???"と表示される。
		LastModified	M	ダウンロード時の更新時にこの情報をもとに判定する。この情報がない場合は、無条件に更新する。
		PackageURL	M	この情報をもとにアプリケーションのダウンロードを行う。
		AppSize	M	この情報をもとにアプリケーションの保存領域を確保する。
		SPSize	O	この情報をもとにスクラッチパッドの領域を確保する。この情報がない場合はスクラッチパッドは使用できない。
		AppClass	M	この情報をもとに起動する i アプリケーションのクラスを決定する。
		AppParam	O	メインクラスの起動パラメータ。
		ScreenSize	O	この情報をもとに i アプリケーションの画面サイズを決定する。

M:必須　O：オプション

4.3.7　JV-Lite2 のメモリ構成

JV-Lite2 Wireless Edition は、簡易ウィンドウシステム WAVE のメモリ管理領域内でで動作するため、WAVE 上で動作するブラウザ（Compact NetFront）など、他のアプリケーションとメモリを共有する。JV-Lite2 が使用するメモリ空間は以下の 3 種類である。

- Java ヒープ領域

  Java ヒープ領域は、JavaVM 上で動作する Java オブジェクトやクラスのインスタンスなどが配置される領域であり、完全に JavaVM だけが管理する。この領域は、JavaVM 起動時に WAVE ヒープ領域から固定的に確保され、終了時までリサイズや解放はできない。領域の最適値はアプリケーションに大きく依存する。

- Java スレッドスタック領域

  Java スレッドスタック領域は、個々の Java スレッドが Java フレームの展開先として使用する領域であり、1 つの Java スレッドが WAVE ヒープ領域の 1 領域を占有する。領域は固定長で、どのスレッドに対しても同じ大きさで確保される。領域の最適値はアプリケーションに依存する。スレッドが多数生成されるアプリケーションは、この領域を多数生成し、メモリを圧迫する。

- WAVE ヒープ領域

  WAVE ヒープ領域は、NetFron3 などの他の WAVE アプリケーションと共有になる。JV-Lite2 のシステムクラスライブラリ、およびネイティブメソッドなどで使用するメモリは、ここから確保される。画像イメージやデータファイルなどの展開先として使われる。

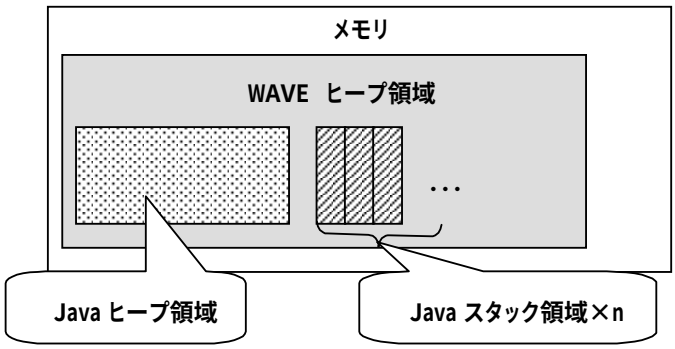

図 35　メモリ使用イメージ

JV-Lite2 Wireless Edition では、VM とクラスライブラリの ROM 化が可能である。これらを ROM 化することで、システムクラスの変更を防ぐとともに、Java アプリケーションの起動時間を短縮することができる。

4.3.8 情報資産の保護

これまで、J2ME 仕様と実装の差異について調査したところ、セキュリティ上影響をあたえる差異はないと考えられる。

JV-Lite2 Wireless Edition では、移植の際に必要に応じてファイルに変更を行うことになるが、その際に変更が加えられていたとしても、最終的に TCK で実行するすべてのテストに合格しなければ販売されないため、基本的には Java、CLDC のセキュリティモデルを実現していることになる。

5 携帯 Java アプリケーションの運用管理におけるセキュリティ課題

携帯電話端末で動作する Java アプリケーションは、OTA(Over The Air)と呼ばれるネットワーク経由でのダウンロード・インストール機構により、Java アプリケーションを格納するインターネットサーバから携帯電話端末に取り込む。Java アプレットと同様に、OTA により取得する信頼のできない Java アプリケーションを携帯電話端末上でいかに安全に実行するかが重要となる。

前章までに見てきたように、低レベルでのセキュリティモデルはサンドボックスモデルにより確立されており、バイトコードの実行時に機器のリソースを破壊しないことが保証されていた。アプリケーションレベルにおいては、使い方によってシステムを破壊したりプライバシーを侵害したりする可能性のある API に関しては、その使用に関する制限がかけられている。API の実行が許可されていない場合には実行の際に例外を起こし、あるいは実行時にユーザに確認を求めることになり、知らない間に実行されるようなことはない。

しかし、こうした Java アプリケーションの実行時におけるセキュリティのチェックとは別に、Java アプリケーションの配布など運用のレベルにおけるセキュリティモデルも検討する必要がある。本章では、携帯電話端末用 Java アプリケーションがインターネットサーバからどのように配布されているのか、また携帯電話端末上ではネットワーク越しの信頼できない Java アプリケーションをどのように管理しているのかについて現状を調査し、どのように運用すべきなのかを検討する。

5.1 携帯 Java アプリケーションの運用

国内通信キャリアにおける現状の Java アプリケーションの配布方法については、API の制限を含め、それぞれ通信キャリアのセキュリティ方針に基づいており、Java アプリケーションの構成方法、およびダウンロード運用についてもそれぞれの特徴がある。

表 52　国内通信キャリアのセキュリティ方針

国内通信キャリア	NTT ドコモ	KDDI（au）	vodafone
Java サービス名	i アプリ	EZ アプリ	V アプリ
アプリケーションの構成	JAR, ADF	KJX	JAR, JAD
アプリケーションの公開方法	公式サイト、または非公式サイトにて公開可能。	選定された企業サイトや公式サイト、または非公式サイトにて公開可能。	コンテンツアグリゲータと呼ばれる公式サイトのみにて公開可能。
配布アプリケーションの制限	非公式サイトからのアプリケーションでは、トラステッド i アプリは実行できない。	非公式サイトからのアプリケーションでは、セキュリティ上利用できない API がある。	非公式サイトを立てることはできず、公式サイトからのアプリケーションには制限がない。

携帯 Java アプリケーションは、一般的には Java のアプリケーション本体（JAR ファイル）と、その Java アプリケーションに関する情報および説明が記述されたアプリケーション記述子と呼ばれるテキストファイルからなる。アプリケーション記述子は、関連付けられた Java のアプリケーションの情報が属性名とその値の組にして説明されており、Java アプリケーションに関する素性を確認できる項目として利用できる。

ネットワークによる Java アプリケーション配布において、セキュリティ上最も重要となるのが、アプリケーション管理ソフトウェア（Application Management Software）と呼ばれる機能である。アプリケーション管理ソフトウェアは、J2SE におけるセキュリティマネージャを代替するものである。これは Java アプリケーションのダウンロード、インストール、検査、起動、更新、削除を行うものであり、国内各通信キャリアの携帯電話端末 Java 環境には JAM（Java Application Manager）と呼ばれるネイティブな機能として実装されている。

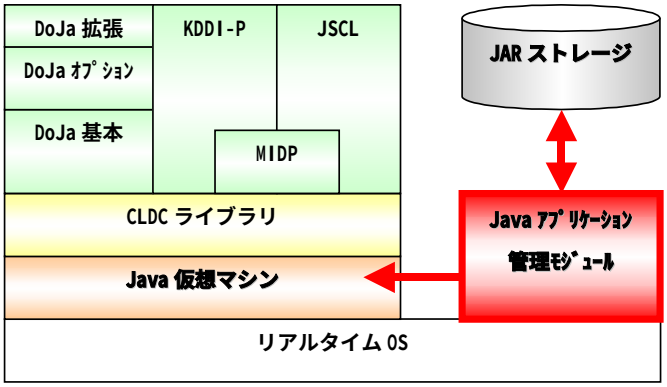

図 36　携帯電話用 Java の構造図

アプリケーション管理ソフトウェアは、Java アプリケーションのダウンロードに際して、まず関連するアプリケーション記述子を参照する。これにより、対象とする Java アプリケーションを実行しないまでも、自身の携帯電話端末にダウンロードしても問題ないアプリケーションであるかどうかなどを事前に判断できる。このように、アプリケーション記述子のメタ情報の内容に依存したセキュリティモデルになっている。

以降では、各国内通信キャリアにおいて Java アプリケーションが現状どのように管理され、配布されているのかを示す。

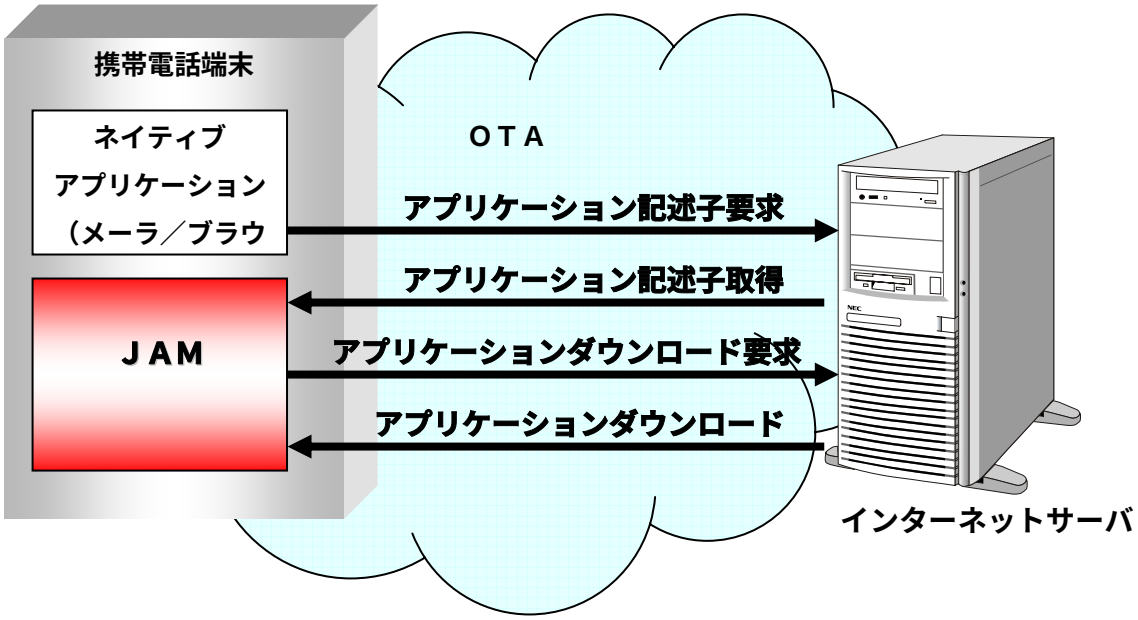

図 37　OTA でのアプリケーションダウンロード手順

5.1.1 NTT ドコモの運用

（1）配布

NTT ドコモの携帯電話端末で動作する i アプリは、NTT ドコモが認定した公式サイトはもちろんのこと、非公式な任意のサイトでも自由に公開することができる。

当初のセキュリティ方針は、公開された API は公式サイトでも非公式サイトでもすべて利用できるというものであったが、これは個人情報などの携帯電話端末のリソースにアクセスしたり、ネイティブな機能を呼び出したりするような API は実装（公開）していないということでもあった。i アプリを構成しているプリミティブとなる API 自身がネイティブのリソースをアクセスしたり、破壊したりできないため、必然的にセキュリティ上問題があるアプリケーションを作成することはできなかった。しかし、最近では、公式サイトからの配布という制限のもとにネイティブのリソースを利用する API が公開されるようになっている。

（2）管理

i アプリは、Java プログラムをコンパイルしたクラスファイルにアプリケーションで利用するイメージや音声などのリソースファイルをまとめた JAR ファイルと、アプリケーション記述子である ADF からなる。ADF は、i アプリに関する属性を記述したファイル（拡張子は「.jam」）であり、Java プログラムと一緒に配布される。

アプリケーション管理ソフトウェアである JAM は、ADF を参照して記述されている各項目のチェックを行い、i アプリを携帯電話端末に正しくダウンロードできるかどうか判定し、i アプリのダウンロード、インストール、更新、削除などの管理を行う。

表 53に、i アプリで指定可能な ADF の属性の一覧を示す。

JAM では、ダウンロードする Java アプリケーションの名前を示す AppName 属性、サイズを記述する AppSize 属性等により、JAR ファイルとの簡単な比較による検証ができる。そして、携帯電話端末のネイティブなリソースにアクセスすることが可能なトラステッド i アプリに対する実行許可の設定なども ADF によって指定できる。ネイティブ機能と連携するようなトラステッド i アプリであっても、ADF への指定によって動作時に例外を発生させることが可能になっている。また、ADF にネイティブなリソースへのアクセス許可指定があった場合でも、実行時にはユーザへの確認が行われるため、十分にセキュリティが考慮された設計になっている。

表 53　i アプリの ADF 属性

属性名	内容	種別
AppName	i アプリ名（最大 16 バイト）	必須
AppVer	i アプリのバージョン番号（最大 10 バイト）	オプション
PackageURL	i アプリの JAR ファイルが配置されている URL（最大 255 バイト）	必須
AppSize	JAR ファイルのサイズ	必須
AppClass:	i アプリの起動に使われるメインクラス名（フルパッケージ名付き・最大 255 バイト）	必須
AppParam	メインクラスの起動パラメータ（最大 255 バイト）	オプション
ConfigurationVer (KvmVer)	J2ME コンフィグレーションバージョン	オプション
ProfileVer	i アプリ実行環境のプロファイルバージョン	オプション
SPsize	スクラッチパッドのサイズ	オプション
LastModified	i アプリの最終変更日時	必須
UseNetwork	ネットワーク機能を使用する場合に通信方式を指定	オプション
TargetDevice	ターゲットとする携帯電話の機種名を指定（最大 128 バイト）	オプション
LaunchAt	i アプリに自動起動を行わせる場合、その起動タイミングを指定	オプション
MyConcierge	待ち受けアプリケーションであることを宣言	オプション
UseTelephone	声通話機能を利用する i アプリであることを宣言	オプション
UseBrowser	ブラウザ機能を利用する i アプリであることを宣言	オプション
LaunchByMail	i アプリがメールからの起動を許可することを宣言	オプション
LaunchByBrowser	i アプリがブラウザからの起動を許可することを宣言	オプション
AllowPushBy	i アプリが赤外線ポートからの vTrigger オブジェクト受信による起動を許可することを宣言	オプション
AppTrace	i アプリ開発のフェーズにおける携帯電話上での簡易なデバッグ機能	オプション
DrawArea	i アプリが前提としている画面サイズ	オプション
GetSysInfo	i アプリが携帯電話のアイコン情報を参照することを宣言	オプション
LaunchApp	アプリケーション連携により他の i アプリを起動（連携モードまたはランチャーモード）する i アプリであることを宣言	オプション
LaunchByApp	アプリケーション連携により他の i アプリから起動されようとする場合の動作を制御	オプション
AccessUserInfo	アプリケーション連携により携帯電話のユーザデータ領域にアクセスする i アプリであることを宣言	オプション
GetUtn	この i アプリが携帯電話の個体識別情報を参照することを宣言	オプション
IletPreserve	ダウンロード即起動 i アプリが自身の携帯電話への保存を禁止することを宣言	オプション

GetUtn 属性に terminalid や userid を指定することで、アプリケーションをダウンロードする際に固体識別番号を取得することができるが、この場合でも必ずユーザの確認が必要となる。

5.1.2 KDDI(au)の運用

（1）配布

KDDI(au)の携帯電話端末で動作する EZ アプリは、KDDI(au)によって認定された公式コンテンツプロバイダで公開することで配布することができる。また、非公式のサイトを立てることも許されており、EZ アプリを自由に公開することもできる。しかし、非公式サイトからダウンロードした EZ アプリがセキュリティ上問題となる可能性のある KDDI-P の拡張 API を利用している場合、アプリケーションの実行時に例外が発生し、動作させることができなくなっている。位置情報の取得等が可能な com.kddi.system パッケージ配下のクラスのほとんどはこの制限のために利用できないようになっており、一般の開発には、CLDC 1.0 のコア API、MIDP 1.0 プロファイル、そして KDDI-P のセキュリティ上保護された API のみが利用できる。

KDDI-P で提供されている各 API には、一般には公開されていないが、それぞれに対してセキュリティレベルが与えられているようである。一部の公開文書では、セキュリティレベル B、C といった部分的な記述はあるが、本調査ではその内容を確認することはできなかった。公式のコンテンツプロバイダなどの選定企業にはすべての API が公開されているが、非公式サイトからアプリケーションを配布する場合の一般の開発者には、そのような情報は開示されていない。したがって、一般の開発者は API の利用に制限がかけられ、セキュリティ上問題のない API のみから作成された安全な Java アプリケーションのみが配布可能となる。

こうした運用により、非公式サイトに公開されている EZ アプリからは、個人情報を参照したり持ち出したりするようなセキュリティ上問題のあるアプリケーションを排除することができる。また、KDDI(au)に認定された公式コンテンツプロバイダなどが作成した公式のアプリケーションは、事前の審査によりアプリケーションに問題があれば配布の前にチェックされることになる。

非公式サイトからダウンロードした EZ アプリはセキュリティ上問題の可能性がある API が利用されていないことが保証され、公式サイトからダウンロードした EZ アプリは KDDI(au)によって審査済のアプリケーションであることが保証されることで、セキュリティが確保されている。

（2）管理

EZ アプリは、アプリケーション本体の JAR ファイルとアプリケーション記述子である JAD（Java Application Descriptor）ファイルを一つにまとめた KJX（KDDI Java eXtension）ファイルである。

JAD ファイルは、EZ アプリに関する属性を記述したテキストファイル（拡張子は「.jad」）である。JAD ファイルに定義されている属性名は MIDP 仕様に準拠しており、さらに KDDI(au)が独自に拡張した属性も定義されている。JAD と ADF と比べると属性名や記述の構文などは異なるが、アプリケーション記述子としての役割は同様であり、対象となる EZ アプリのサイズや振る舞いなどの情報が属性名と値の組にして記述される。

アプリケーション本体の JAR ファイルは、1 つ以上の MIDlet を含むクラスファイル、マ

ニフェストファイル､そしてEZアプリで利用する画像や音声などのリソースファイルをJARファイルの形式にまとめたものである。JARファイルに含められるマニフェストファイルはJAR ファイルのパッケージ情報を記述するもので、JAD ファイルと同等の意味を持つ「MANIFEST.MF」というファイル名のテキストファイルである。JADファイル、マニフェストファイルには､属性名と値をコロン記号で区切って表した組の属性リストが記述される。

KDDI(au)の携帯電話端末にもJAMと呼ばれるアプリケーション管理ソフトウェアが実装されており、JADファイルを利用してEZアプリを検査、および管理している。MIDP仕様においてJAMは､マニフェストファイルとアプリケーション記述子の共通の属性の値が一致するかどうかのチェックも行う。またKJXファイルには、CRCチェックサムも添付されているので、ファイルの破損や変更のチェックもできるようになっている。

表 54に､EZアプリで指定可能なJADの属性の一覧を示す｡表内のMANIFEST項目は、マニフェストファイルで指定する属性を示している。

表 54　EZアプリのMIDlet属性

属性名	内容	種別	MANIFEST
MIDlet-Name	EZアプリ名	必須	必須
MIDlet-Version	EZアプリのバージョン情報	必須	必須
MIDlet-Vendor	EZアプリの提供者	必須	必須
MIDlet-Icon	EZアプリを表すのに使うJARファイル中の画像ファイル名	―	オプション
MIDlet-Description	EZアプリの詳細説明	―	オプション
MIDlet-Info-URL	EZ アプリのプロパティからHTMLドキュメントへ遷移	オプション	オプション
MIDlet-<n>	JARファイル中､カンマで区切られたn番目のMIDletの名前、アイコン、クラス、名前	オプション	必須
MIDlet-Jar-URL	EZアプリをパッケージしたJARファイル名	必須	―
MIDlet-Jar-Size	JARファイルのファイルサイズ	必須	―
MIDlet-Data-Size	EZ アプリがデータストレージを使用する場合のみ、その使用量（単位バイト）を記述	オプション	オプション
MicroEdition-Profile	EZ アプリが動作可能なプロファイル	オプション	必須
MicroEdition-Configuration	EZ アプリが動作可能なコンフィグレーションを記述	オプション	必須
MIDlet-X-Copyright	著作権フラグ、オプションが指定されていない場合、著作権保護の必要なしと解釈	オプション	―
MIDlet-X-AutoLaunch	コンテンツ作成者が指定するEZアプリの自動起動設定条件	オプション	―
MIDlet-X-AllowURL-<n>	EZアプリからHTTP通信機能を利用する場合に、HTTP通信時に接続を許可するURLを記述	オプション	―

5.1.3 vodafone の運用

（1）配布

vodafone の携帯電話端末で動作する V アプリは vodafone が認定したサイト運営者であるコンテンツアグリゲータの公式サイトにのみ登録される。非公式なサイトを勝手に立てることは許されておらず、V アプリの公開には一定の審査が行われ、その審査にパスしないと公開することができない。審査では、携帯電話端末内の個人情報を参照して配布するような悪意のあるアプリケーションや公序良俗に反するアプリケーションなどはチェックされ、排除される。

公開されている JSCL プロファイルの API の中には携帯電話端末内の個人情報を取得できるものもあるが、非公式サイトからは Java アプリケーションを公開できず、V アプリを配布できるサーバをコンテンツアグリゲータ企業のサイトに限定することと、審査により問題があるアプリケーションが配布されることがないことにより、セキュリティが確保されている。

（2）管理

V アプリは、アプリケーション本体の JAR ファイルとアプリケーション記述子である JAD ファイルからなる。

JAD ファイルは、V アプリに関する属性を記述したテキストファイル（拡張子は「.jad」）である。JAD ファイルに定義されている属性名は MIDP 仕様に準拠しており、さらに vodafone が独自に拡張した属性が追加されている。対象となる V アプリのサイズや振る舞いなどの情報が属性名と値の組にして記述される。

アプリケーション本体の JAR ファイルは、1 つ以上の MIDlet を含むクラスファイル、マニフェストファイル（拡張子は「mf」）、リソースファイルを JAR ファイルの形式にまとめたものである。JAR ファイルに含められるマニフェストファイルは JAR ファイルのパッケージ情報を記述するもので、EZ アプリのマニフェストファイルと同様である。JAD ファイル、マニフェストファイルには、属性名と値をコロン記号で区切って表した組の属性リストが記述される。

vodafone の携帯電話端末にもアプリケーション管理ソフトウェア JAM がネイティブ機能として実装されており、アプリケーション記述子である JAD をチェックして、V アプリケーションの管理を行っている。表 55に、V アプリで指定可能な JAD の属性の一覧を示す。表内の MANIFEST 項目は、マニフェストファイルで指定する属性を示している。

ネットワークにアクセスする機能を持った V アプリの場合、JAD ファイルもしくは Manifest ファイルに「MIDlet-Network:Y」の記述がないと、実行時に SecurityException が発生する。また、JAD ファイルもしくはマニフェストファイルに「MIDlet-Network:Y」の記述があった場合でも、ユーザが端末上でネットワークアクセスを禁止するよう設定している場合には、実行時に IOException が発生する。

MIDlet-Application-Security は、V アプリのセキュリティレベルを設定する属性である。「MIDlet-Application-Security:Y」であれば、表 48にあるセキュリティ対象クラスメソッドを起動することができる。「MIDlet-Application-Security:N」の指定あるいは属性がない

場合は、セキュリティ対象クラスメソッドが起動できなくなっている。

表 55　V アプリの MIDlet 属性

属性名	内容	種別	MANIFEST
MIDlet-Name	V アプリ名	必須	必須
MIDlet-Version	V アプリのバージョン	必須	必須
MIDlet-Vendor	V アプリのベンダ名	必須	必須
MIDlet-Icon	V アプリを表すのに使う JAR ファイル中の PNG ファイルの名前	オプション	オプション
MIDlet-Description	V アプリの詳細説明	オプション	オプション
MIDlet-Info-URL	V アプリのプロパティから HTML ドキュメントへ繊維	オプション	オプション
MIDlet-<n>	JAR ファイル中､カンマで区切られた n 番目の MIDlet の名前、アイコン、クラス、名前	オプション	必須
MIDlet-Jar-URL	JAR ファイルを指す URL	必須	－
MIDlet-Jar-Size	JAR ファイルのサイズ	必須	－
MIDlet-Data-Size	永続記憶域のサイズ	オプション	オプション
MicroEdition-Profile	MIDlet が必要とする J2ME のプロファイル	オプション	必須
MicroEdition-Configuration	MIDlet が必要とする J2ME のコンフィグレーション	オプション	必須
MIDlet-Network	HttpConnection を利用するか否かを指定	オプション	－
MIDlet-Save	V アプリを周辺機器に転送可能か否かを指定	オプション	－
MIDlet-Copyright	著作権の有無を指定	オプション	－
MIDlet-Resource	待受型にするか否かを指定	オプション	－
MIDlet-OCL	JSCL ライブラリを指定	必須	－
MIDlet-Application-Range	高精細液晶サイズの V アプリを作成するのに使用	オプション	－
MIDlet-Serial	シリアル制御を利用するか否かを指定	オプション	－
MIDlet-RemoteControl	赤外線リモコン機能を使用するか否かを指定	オプション	－
MIDlet-Application-Security	端末の内部情報取得機能を使用するか否かを指定	オプション	－

5.2 公開鍵方式によるアプリケーション配布

これまでに見てきたように､現在の国内通信キャリアの携帯電話端末における Java 環境のセキュリティモデルは、サンドボックスと拡張プロファイルの API 制限、配布アプリケーションの事前審査、そしてアプリケーション記述子を利用したアプリケーション管理ソフトウェアにより確立されている。

アプリケーション管理ソフトウェアは、アプリケーション記述子をもとにダウンロードしようとする Java アプリケーションを簡易的に認証する。Java アプリケーションのダウンロードの可否を決定し、その後のインストール、実行、更新、削除までの一連の管理を担う。Java アプリケーションを取り込むという入り口の部分に位置するアプリケーション管理ソフトウェアは、携帯電話端末のセキュリティを保護する上で非常に重要であり、その果たす役割も大きい。

MIDP 2.0 仕様では、プロテクションドメインと公開鍵方式による署名を導入して「信頼できる MIDlet スイート」を配布する仕様が規定されている。MIDP 1.0 での Java 環境のセキュリティモデルは、基本的にはサンドボックスモデルでしかなかった。MIDP 2.0 仕様では、アプリケーション記述子にパーミッションからなるプロテクションドメインと証明書およびアプリケーションへの署名を設定する。署名されていない MIDlet スイートは、素性がはっきりしない「信頼できない MIDlet スイート」となる。

プロテクションドメインはパーミッションのリストであり、パーミッションによりセキュリティ上問題となる可能性のある API の実行を許可したり、API 実行時にユーザに確認させるインタラクティブモードを指定したりすることができる。

表 56にパーミッションの一覧を示す。個々のパーミッションにより、ネットワーク接続や個人情報を取得するような携帯電話端末のネイティブなリソースにアクセスする API に対する実行許可を与えることができる。パーミッションで指定できるインタラクティブモードには、blanket, session, oneshot のモードがある。

表 56　パーミッションの概要

パーミッション	モード	内容
Allowed	-	ユーザに対して実行の許可の確認を要求しないでアプリケーションを実行できる
User	blanket	アンインストールされるかパーミッションが変更されるまで、実行の許可の確認を要求しないでアプリケーションを実行できる。アプリケーションのインストール時にだけユーザに対して実行の許可の確認を要求する。
	session	MIDlet スイートが終了するまで、実行の許可の確認を要求しないでアプリケーションを実行できる。保護されている API や機能の最初の起動時にユーザに対して実行の許可の確認を要求する。
	oneshot	保護されている API や機能を起動するたびに毎回ユーザに対して実行の許可の確認を行う。

表 57は、MIDP 2.0 で事前に定義された MIDlet 属性である。MIDP 2.0 のアプリケーション記述子の新たな属性として MIDlet-Permissions と MIDlet-Permissions-Opt が定義されているが、これらはプロテクションドメインを記述するための属性であり、パーミッションのリストを記述するものである。

表 57　MIDP 2.0 の MIDlet 属性

属性名	内容	種別	MANIFEST
MIDlet-Name	MIDlet スイートの識別子名	必須	必須
MIDlet-Version	MIDlet スイートのバージョン	必須	必須
MIDlet-Vendor	MIDlet スイートを提供する組織	必須	必須
MIDlet-Icon	MIDlet スイートを表すのに使う JAR ファイル中の PNG ファイルの名前	オプション	オプション
MIDlet-Description	MIDlet スイートの説明	オプション	オプション
MIDlet-Info-URL	より詳細な MIDlet スイートの説明がある URL	オプション	オプション
MIDlet-<n>	JAR ファイル中、カンマで区切られた n 番目の MIDlet の名前、アイコン、クラス、名前	オプション	必須
MIDlet-Jar-URL	JAR ファイルをロードする URL	必須	オプション
MIDlet-Jar-Size	JAR ファイルのバイト数	必須	オプション
MIDlet-Data-Size	MIDlet により要求された固定データ用の最小バイト数（デフォルトは、0）	オプション	オプション
MicroEdition-Profile	MIDlet が必要とする J2ME のプロファイル	オプション	必須
MicroEdition-Configuration	MIDlet が必要とする J2ME のコンフィグレーション	オプション	必須
MIDlet-Permissions	MIDlet スイート内のクリティカルな機能に対するパーミッション	オプション	オプション
MIDlet-Permissions-Opt	MIDlet スイート内のクリティカルでない機能に対するパーミッション	オプション	オプション
MIDlet-Push-<n>	MIDlet スイートをプッシュ配信として登録する	オプション	オプション
MIDlet-Install-Notify	MIDlet スイートのインストール結果を POST する先の URL	オプション	オプション
MIDlet-Delete-Notify	MIDlet スイートの削除結果を POST する先の URL	オプション	オプション
MIDlet-Delete-Confirm	MIDlet スイートの削除時に確認されるメッセージ	オプション	オプション

X.509 の証明書を利用した署名を利用するには、証明書と署名についての記述がアプリケーション記述子に必要となる。証明書は、MIDlet-Certificate-<n>-<m>という属性に証明書を Base64 でエンコードしたテキストを記述する。署名は、MIDlet-Jar-RSA-SHA1 という属性に、MIDlet と MANIFEST が含まれた JAR ファイルをハッシュ関数 SHA-1 と公開鍵暗号 RSA を用いて署名したものを Base64 でエンコードして記述する。

X.509 証明書を利用した公開鍵方式による Java アプリケーションの配布では、携帯電話端末側で秘密鍵やルート証明書を保持する必要があるが、十分なセキュリティを確保できる。

5.3 情報資産の保護

信頼できないネットワーク越しにダウンロードした信頼できない Java アプリケーションを携帯電話端末上でいかにして安全に実行させるかは、ダウンロードした Java アプリケーションの出所、素性がはっきりしていることと、その Java アプリケーションが第三者などにより不正に改ざん等されていないことが保証されることで解決される。

国内通信キャリアの Java アプリケーションのダウンロードサービスでは、携帯電話端末の個人情報を参照するようなネイティブなリソースを扱うといったセキュリティ上問題が生じる可能性のある Java アプリケーションに対しては、配布の事前に各キャリアに対する申請が必要となる。なおかつ、そうした Java アプリケーションは非公式サイトからは配布できず、公式サイトからのみ配布が許可される。

これにより、公式サイトから配布される Java アプリケーションの提供元は特定され、たとえ審査が不十分で Java アプリケーションが予期せぬ動作を起こした場合でも、責任の所在は明確になる。不正な Java アプリケーションや悪意のある Java アプリケーションが配布されないだけでなく、抑制する効果を持った運用のセキュリティモデルになっている。

国内通信キャリアにおいては、信頼できるネットワーク越しに提供元が明らかな Java アプリケーションを配布することでセキュリティを確保している。

また、将来的には MIDP 2.0 の仕様で示されているプロテクションドメインを利用した公開鍵方式による運用がなされていくものと思われる。アプリケーション記述子にプロテクションドメインによる API 単位の保護を指定でき、提供元の証明書と Java アプリケーション本体に対する署名の添付が可能になる。これにより、セキュリティ上問題を生じる可能性がある API の実行時には、ユーザの確認を要請することができ、ダウンロード対象とする Java アプリケーションの素性が明らかにされ、Java アプリケーションの改ざんについても検査が可能になる。これにより信頼できないネットワークからのダウンロードであっても信頼できる Java アプリケーションを安全に実行できるセキュリティモデルとなる。

6 まとめ

本報告書では、携帯電話用 Java について、セキュリティの観点から以下の調査・分析を行った。

- J2ME 仕様におけるセキュリティ課題
- Java バーチャルマシン(VM)のセキュリティモデルおよび機能
- 携帯 Java アプリケーションの運用管理におけるセキュリティ課題

6.1 調査結果

3章では、J2ME の仕様において策定されている API を対象として、どのようにして Java アプリケーションのセキュリティが実現されているのかを調査した結果を示した。

J2ME CLDC コンフィグレーションでは、ファイル操作を行うクラスをサポートしない等、セキュリティ上不完全なクラスやメソッドはサポートしないことで、高いセキュリティを実現している。しかし、CLDC は小型の組み込み機器用に搭載すべき最低限の機能のみを規定しているため、利便性に欠ける。そこで実際には、NTT ドコモ、KDDI、vodafone 各社とも独自のプロファイルを CLDC 上に動作させることで、ユーザインタフェース、ネットワーク機能などを利用できるようにしている。これらの独自プロファイルでは、携帯電話端末に登録されたメールアドレス・電話番号等の情報資産へアクセスする API を提供しており、情報資産を参照するアプリケーションの開発も可能である。しかし、このようなアプリケーションは、公式サイトからのみダウンロードを許可し、実行時には必ず利用者の確認を取る仕組みになっており、運用と組み合わせることでセキュリティの確保が行われている。つまり、利用者の知らないうちに携帯電話端末に登録されている情報資産が漏洩することはない。

4章では、3 種類の JavaVM を対象として低レベルのセキュリティモデルを調査した結果を示した。

各 JavaVM とも、J2ME 仕様に準拠しているため、ユーザ定義のクラスローダをサポートしないことと、2 フェーズで行うクラスファイルベリファイアにより、サンドボックスモデルを実現し、セキュリティを確保している。クラスファイルベリファイアが機能し、Java アプリケーションが他のメモリへ影響を与えないようエラークラスを呼び出して安全に終了することや、サポートしていない API を利用した Java アプリケーションが実行できないことが、実験からも確認できた。つまり、不正な Java アプリケーションによって、携帯電話端末に登録されている情報資産を破壊することはできない。

しかし、システムクラスのなりすましやクラスファイルの改ざんといったセキュリティ上の問題が残る。システムクラスのなりすましに関しては、VM メーカや端末機メーカでの対策が採られている。VM の実験では、クラスファイルのバイトコード中に含まれるデータや演算命令セットを改ざんし、それを実行させることが可能であることも確認できた。VM の

レベルではデータやプログラムが改ざんされていないかを確認することはできないため、上位レベルでの対応が必要となる。

そこで、上位レベルでの対応として、各国内通信キャリアが運用上どのようにセキュリティを確保しているのかを調査し、その結果を5章において示した。

3章の調査結果にあるように、各社で定義したプロファイルには、携帯電話端末上の情報資産へアクセスするAPIが提供されている。しかし、各社とも、このようなAPIを利用したセキュリティ上問題が生じる可能性のあるJavaアプリケーションに対しては、登録の際に各キャリアに対する申請を必要としている。なおかつ、非公式サイトからは配布できず、公式サイトからのみ配布が許可される。これにより、公式サイトから配布されるJavaアプリケーションの提供元は特定され、たとえ審査が不十分でJavaアプリケーションが予期せぬ動作を起こした場合でも責任の所在は明確になる。不正なJavaアプリケーションや悪意のあるJavaアプリケーションが配布されないだけでなく、抑制する効果を持った運用のセキュリティモデルになっている。

データやプログラムの改ざんへの対策として、MIDP 2.0では署名付きのアプリケーション配布に対応している。これにより、ダウンロード対象とするJavaアプリケーションの素性が明らかにされ、Javaアプリケーションの改ざんについても検査が可能になる。

6.2 考察

J2ME仕様におけるセキュリティ課題は、信頼できないネットワークからダウンロードした信頼できないJavaアプリケーションをリソースの制限された端末機器でいかに安全に実行させるかである。制限されたリソースの中でセキュリティを実現するためには、仮想マシンなどの低レベルでのセキュリティ設計を見直し、アプリケーションレベルではセキュリティ上問題になる可能性があるAPIの制限等によるサンドボックスモデルが有効であった。サンドボックスモデルのポリシーは、すべてのJavaアプリケーションは信頼できないものとして、保護されたAPI制限のもとで実行するというものである。当初、このサンドボックスモデルは、携帯電話端末においても有効であった。

しかし、携帯電話端末の急速な高機能化に伴い、高度なサービスを提供できるようになった現在では、サンドボックスだけによるセキュリティモデルでは、あまりにも制限が強すぎてその利便性を活かすことができなくなってしまっている。すなわち、現在では、高機能な携帯電話の利便性を活かしながら、しかも制限されたリソース内で、いかにJavaアプリケーションを安全に実行させるかが、携帯電話端末におけるセキュリティ課題となっている。

携帯電話端末のセキュリティ課題は、低レベルのセキュリティ、アプリケーションレベルのセキュリティ、エンド・ツー・エンドのセキュリティとそれぞれのレベルのセキュリティ設計、および再設計により解決されている。

国内通信キャリアでは、セキュリティ上問題となる可能性がある利便性の高いJavaアプリ

ケーションは、公式サイトからでしかダウンロードすることができない。信頼できるネットワークから、通信キャリアによって提供元が明確にされている Java アプリケーションのみをダウンロードさせるという運用である。こうした運用により、携帯電話端末の利便性を損なうことなく、Java アプリケーションを安全に実行することが保証され、悪意のある不正な Java アプリケーションは排除される。

その一方で、国内通信キャリアの現在の運用は、一般の開発者に対して携帯 Java アプリケーションの開発を制限しており、より一層の発展が見込まれる携帯電話端末の Java 環境を利用したビジネスの機会を抑制してしまっている。本来は、信頼できないネットワークからでも信頼できる Java アプリケーションを自由にダウンロードでき、安全に実行できる環境が望まれるはずである。

そうした要求に対する 1 つの解が MIDP 2.0 のセキュリティ仕様で与えられている。そこでは、セキュリティ上問題の可能性のある API を利用した Java アプリケーションに対しては、プロテクションドメインと証明書を利用した公開鍵方式によって配布する方法が規定されている。公開鍵方式による Java アプリケーションの配布は、信頼できないネットワークからでも信頼できるアプリケーションをダウンロードし、安全に実行することを可能にする。携帯電話端末内のアプリケーション管理ソフトウェアは、アプリケーション記述子に添付された証明書と署名により、その素性を確認でき、改ざんのチェックもできるため、信頼性、整合性、完全性が確保された信頼できる Java アプリケーションを安全に実行できる。

今後も Java 搭載携帯電話は、より高度化し益々普及してゆくものと思われる。それに伴い、各通信キャリアや機器メーカは独自の機能を追加していくことが予想される。より強いセキュリティを確保するためには、PKI 基盤の導入なども必要となるだろう。

付録 A　参考文献

携帯電話 Java 環境におけるセキュリティ技術に関して調査した文献を以下に示す。

下記のリストは、[文献番号] 著者名／発行元,「文献名」, 発行日付、の順に記載している。論文、Web 情報については、文献名に続き、所在 URL を記載する。ただし、発表日付が明記されていないものに関しては、そのページデータの最終更新年を記載している。また、URL は 2003 年 10 月現在のものである。

書式

[文献番号]　　著者名／発行元
　　「文献名」
　　所在 URL（論文、Web 情報）
　　発行日付

[A1]　NTT ドコモ
　　「i モード対応 Java コンテンツ開発ガイド～詳細編～1.1 版」,
　　2001/06/22

[A2]　NTT ドコモ
　　「i モード対応 Java コンテンツ開発ガイド～API リファレンス編～第 1.1 版」
　　2001/06/22

[A3]　NTT ドコモ
　　「i アプリコンテンツ開発ガイド　for 504i～詳細編～第 2.2 版」
　　2003/07/24

[A4]　NTT ドコモ
　　「i アプリコンテンツ開発ガイド　for 504i～i アプリオプション・拡張編～2.0 版 」
　　2002/11/12

[A5]　NTT ドコモ
　　「API リファレンス（i アプリ基本 API 編）1.0 版」
　　2003/07/24

[A6]　NTT ドコモ
　　「API リファレンス（i アプリオプション・拡張 API 編）2.0 版」
　　2002/11/12

[A7]　NTT ドコモ
　　「各機種オプション API・拡張 API 実装状況 2.3 版」
　　2003/07/24

[A8]　NTT ドコモ
　　「i アプリコンテンツ開発ガイド　for DoJa-3.0～詳細編～第 1.00 版」
　　2003/04/17

[A9] NTT ドコモ
「i アプリコンテンツ開発ガイド　for DoJa-3.0～i アプリオプション・i アプリ拡張編～1.0 版」
2003/04/17
[A10] NTT ドコモ
「i アプリコンテンツ開発ガイド　for DoJa-3.0～API リファレンス編～第 1.0 版」
2003/04/17
[A11] NTT ドコモ
「i アプリコンテンツ開発ガイド　for DoJa-3.0～オプション・拡張 API リファレンス編～第 1.0 版」
2003/04/17
[A12] NTT ドコモ
「i アプリコンテンツ開発ガイド　for DoJa-3.0～各機種オプション API・拡張 API 実装状況～第 1.0 版」
2003/08/06
[A13] NTT ドコモ
「DoJa-3.0API　iappli Development Kit　ユーザーズガイド」
2003/04/22
[A14] KDDI
「EZ アプリ(Java[TM])プログラミングガイド Version-1.1」
2003/07/11
[A15] KDDI
「EZweb 仕様書　KJX 作成ツール取扱説明書　Version-1.1」
2002/5/10
[A16] KDDI
「EZweb 仕様書　ezplus エミュレータ取扱説明書　Version-1.0」
2002/5/10
[A17] vodafone
「J-PHONE Java™アプリ開発ガイド Version1.2.0」
2003/2/10
[A18] vodafone
「ボーダフォンライブ！向け V アプリ開発ガイド［概要編］ 1.0.0」
2003/10/1
[A19] vodafone
「ボーダフォンライブ！向け V アプリ開発ガイド［メディア編］ 1.0.0」
2003/10/1
[A20] vodafone
「ボーダフォンライブ！向け V アプリ開発ガイド［シリアル制御編］ 1.0.0」
2003/10/1

[A21] vodafone
「ボーダフォンライブ！向けＶアプリ開発ガイド［Tips 編］」
2003/10/1
[A22] vodafone
「ボーダフォンライブ！向けＶアプリ開発ガイド［開発編］」
2003/10/1
[A23] vodafone
「開発ガイド新旧用語対照表」
[A24] vodafone
「J-PHONE Java アプリプログラミングガイド」
2002/05/16
[A25] vodafone
「J-SKY Application Emulator　ユーザーズガイド　Version1.0.1」
2003/3/17
[A26] vodafone
「V-appli Emulator for JSCL 1.0.1 ユーザーズガイド」
2003/10/01
[A27] Sun Microsystems
「Connected,Limited Device Configuration 仕様　バージョン 1.0（日本語版）」
2000/7/31
[A28] Sun Microsystems
「J2ME Building Blocks for Mobile Devices－White Paper on KVM and the Connected,Limited Device Configuration（CLDC）（KVM ホワイトペーパー）」
2000/5/19
[A29] Sun Microsystems
「KVM ポーティングガイド－KVM1.0」
2000/7/31
[A30] Sun Microsystems
「Porting MIDP - MIDP Reference Implementation, Version 2.0 FCS」
[A31] Sun Microsystems
「Using MIDP - MIDP Reference Implementation, Version 2.0 FCS」
[A32] Sun Microsystems
「ユーザーズガイド－Wireless Toolkit,Version1.0.4　Java™2 Platform,Micro Edition」
2002/6
[A33] アプリックス
「CLDC1.0 仕様 microJBlend デバイスエミュレータリリースノート」
2003/10/21

[A34] ACCESS
「JV-Lite2 Wireless Edition Ver1.0 技術資料」
2003/10
[A35] Java Community Process
「CLDC Library API Specification 1.0」
[A36] Java Community Process
「CLDC 1.1」（javadoc）
[A37] NTT ドコモ
「DoJa-1.0 API」（javadoc）
[A38] NTT ドコモ
「DoJa-2.0 API」（javadoc）
[A39] NTT ドコモ
「DoJa-2.0 オプション API」（javadoc）
[A40] NTT ドコモ
「DoJa-3.0 API」（javadoc）
[A41] NTT ドコモ
「DoJa-3.0 オプション API」（javadoc）
[A42] Java Community Process
「MID Profile」（javadoc）
[A43] Java Community Process
「Mobile Information Device Profile 2.0」（javadoc）
[A44] Sun Microsystems
「JavaTM 2 Platform, Standard Edition, v 1.4.2 API Specification」（javadoc）
[A45] vodafone
「C4 API JSCL 1.0 API」（javadoc）
[A46] vodafone
「JSCL 1.07 API」（javadoc）
[A47] vodafone
「JSCL1.2 API」（javadoc）
[A48] 橋本賢一／ナツメ社
「図解最新テクノロジー 携帯 Java」
2002/12/31
[A49] KDDI
「KDDI Profile Javadoc Document Revision: 2.01.01」（javadoc）
2002/06/21
[A50] 神戸博之、高坂一城／IDG ジャパン
「J2ME プログラミングガイド」
2003/09/09

[A51] Feng Yu、Zhu Jun 著　鷺谷好輝、ゼンテックテクノロジージャパン訳／アスキー
「J2ME ワイヤレス Java プログラミング」
2002/05/11
[A52] インフォシェル／毎日コミュニケーションズ
「ケータイサイト構築完全ガイド」
2002/12/05
[A53] G.マグロー、E.W.フェルテン著　林秀幸訳／トッパン
「JAVA セキュリティ」
1997/08/26
[A54] ティム・リンドホルム、フランク・イェリン著　野崎裕子訳／アジソン・ウェスレイ・パブリッシャーズ・ジャパン
「The Java™仮想マシン仕様」
1997/12/25
[A55] ティム・リンドホルム、フランク・イェリン著　村上雅章訳／ピアソン・エデュケーション
「Java™仮想マシン仕様 第 2 版」
2001/05/25
[A56] Jon Meyer、Troy Downing 著　鷲見豊訳・編著／オライリージャパン
「JAVA バーチャルマシン」
1997/07/28
[A57] イーバレー／技術評論社
「まるごと図解最新組み込み Java がわかる」
2003/05/01
[A58] IDG ジャパン
「JavaWORLD　2003.1 月号
2003/01/01
[A59] Sun Microsystems
「Java 2 Platform, Micro Edition (J2ME)」
http://java.sun.com/j2me
[A60] Sun Microsystems
「J2ME & コンシューマ／組込み向け Java テクノロジ」
http://sdc.sun.co.jp/java/j2me/index.html
[A61] Sun Microsystems
「CLDC」
http://java.sun.com/products/cldc/
[A62] Sun Microsystems
「Mobile Information Device Profile (MIDP) 」
http://java.sun.com/products/midp/

[A63] Sun Microsystems
「Sun Developer NEWS」
http://sdc.sun.co.jp/news/index.html
[A64] Sun Microsystems
「Sun Microsystems - Wireless」
http://sdc.sun.co.jp/java/wireless/index.html
[A65] Sun Microsystems
「Sun Community Source Licensing (SCSL)」
http://wwws.sun.com/software/communitysource/
[A66] Sun Microsystems
「Java™センター　サンプルコード集」
http://javacenter.sun.co.jp/java-sample/
[A67] NTT ドコモ
「ｉアプリコンテンツの作成について」
http://www.nttdocomo.co.jp/p_s/imode/java/
[A68] Java Community Process
「JSR-000037 Mobile Information Device Profile (MIDP) (Final Release)」
http://www.jcp.org/aboutJava/communityprocess/final/jsr037/
[A69] Java Community Process
「JSR-000118 Mobile Information Device Profile 2.0」(Final Release)」
http://www.jcp.org/aboutJava/communityprocess/final/jsr118/
[A70] Sun Microsystems
「Java™ security」
http://java.sun.com/security/
[A71] Sun Microsystems
「セキュリティ」
http://java.sun.com/j2se/1.4/ja/docs/ja/guide/security/
[A72] Java Community Process
「The Java Community Process」
http://www.jcp.org/ja/home/index
[A73] Java Community Process
「JSR 30 J2ME™ Connected, Limited Device Configuration」（CLDC 1.0 仕様）
http://www.jcp.org/ja/jsr/detail?id=30
[A74] Java Community Process
「JSR 139 Connected Limited Device Configuration 1.1」（CLDC 1.1 仕様）
http://www.jcp.org/ja/jsr/detail?id=139
[A75] Java Community Process
「JSR 177 Security and Trust Services API for J2ME™」
http://jcp.org/en/jsr/detail?id=177

[A76] NTT ドコモ
「iMODE」
http://www.nttdocomo.co.jp/p_s/imode/
[A77] NTT ドコモ
「i アプリコンテンツの作成について」
http://www.nttdocomo.co.jp/mc-user/i/java/index.html
[A78] vodafone
「vodafone Developers Support Site」
http://www.dp.j-phone.com/dp/
[A79] i-JADE
「i-JADE Basic」
http://www.zentek.co.jp/jpn/products/mobile/jade/ijade/basic/index.html
[A80] ベクター
「週間ゲーム＆アプリ Live」
http://dev.javalive.jp/
[A81] スパイシーソフト
「アプリ★ゲット V!」
http://appget.com/vf/pc/
[A82] KDDI
「KDDI」
http://www.kddi.com/
[A83] KDDI
「au by KDDI」
http://www.au.kddi.com/
[A84] KDDI
「ezplus 技術情報」
http://www.au.kddi.com/ezfactory/tec/spec/ezplus.html
[A85] アプリックス
「JBlend」
http://www.jblend.com/
[A86] ACCESS
「ACCESS」
http://www.access.co.jp/top.html
[A87] NETGENE
「組込み Java 情報源」
http://www.netgene.co.jp/java/index.html
[A88] ソフトバンク
「特集　携帯向け Java，3 社の違いは？」
http://www.zdnet.co.jp/mobile/0104/25/java.html

[A89] ソフトバンク
「i モード対応 Java を作るのに必要なものは？――NTT ドコモ, 仕様をついに公開」
http://www.zdnet.co.jp/news/0012/27/java.html
[A90] ソフトバンク
「Java アプリ事始め 第 1 回：
J-フォン Java アプリがやってきた！」（実験サンプルソース引用）
http://www.zdnet.co.jp/mobile/0203/08/n_j1.html
[A91] 情報処理振興事業協会（現：情報処理推進機構）
「Java 環境でのウイルスの危険性に関する調査報告書」
http://www.ipa.go.jp/security/fy11/report/contents/virus/rep_java.pdf
[A92] 情報処理振興事業協会（現：情報処理推進機構）
「Java を搭載する携帯端末機器におけるコンピュータウイルスの危険性に関する調査・検討結果報告書」
http://www.ipa.go.jp/security/fy12/contents/virus/report/java-pda.pdf
[A93] 情報処理振興事業協会（現：情報処理推進機構）
「Java 対応携帯電話機の Java ウイルスの危険性に関する調査・検討報告書」
http://www.ipa.go.jp/security/fy13/report/mobile_security/java-keitai.pdf